SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

# DIPLOMSKI RAD

**Goran Antić**

Zagreb, 2013.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

# DIPLOMSKI RAD

Mentor: Student:

Prof. dr. sc. Bojan Jerbić, dipl. ing. Goran Antić

Zagreb, 2013.

Izjavljujem da sam ovaj rad izradio samostalno koristeći stečena znanja tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru prof. dr. sc. Bojanu Jerbiću što mi je dao priliku raditi na ovom radu, te na savjetima koje mi je pružio prije i tijekom izrade rada. Također, zahvaljujem se i asistentu dr. sc. Tomislavu Stipančiću na motivacijskoj potpori i nesebičnoj pomoći na svim poljima gdje mi je bilo potrebno da savladam prepreke. Također, zahvaljujem i dr. sc. Petru Ćurkoviću i Denisu Bašiću na ugodnoj atmosferi u laboratoriju. Zahvaljujem se asistentima, mag. ing. Marku Švaci i mag. ing. Bojanu Šekoranji na informacijama i podacima koje su mi dali na uvid i korištenje prilikom rada u Laboratoriju za projektiranje izradbenih i montažnih sustava, te na nesebičnoj pomoći.

Na kraju, zahvaljujem roditeljima i sestri na razumijevanju prilikom pisanja rada.

Goran Antić

SVEUČILIŠTE U ZAGREBU
**FAKULTET STROJARSTVA I BRODOGRADNJE**
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

| Sveučilište u Zagrebu<br>Fakultet strojarstva i brodogradnje | |
|---|---|
| Datum | Prilog |
| Klasa: | |
| Ur.broj: | |

# DIPLOMSKI ZADATAK

Student: **GORAN ANTIĆ** Mat. br.: 0035169544

Naslov rada na hrvatskom jeziku: **KORISNIČKO SUČELJE ZA UPRAVLJANJE INDUSTRIJSKIM ROBOTOM**

Naslov rada na engleskom jeziku: **USER INTERFACE FOR AN INDUSTRIAL ROBOT CONTROL**

Opis zadatka:

Razvoj suvremenih vizijskih sustava omogućuje nove i nekonvencionalne pristupe upravljanja robotskim sustavima. Primjenom Microsoft Kinect stereo-vizijskog sustava moguće je interpretirati ljudske kretnje te ih potom iskoristiti za interakciju s robotom. U skladu s time potrebno je proučiti tehničke i programske značajke Microsoft Kinect stereo-vizijskog sustava, te razviti programsku podršku za upravljačko korisničko sučelje koje će se koristiti za intuitivno upravljanje robota prilikom rukovanja objektima.

Razvijenu primjenu provjeriti koristeći opremu dostupnu u Laboratoriju za projektiranje izradbenih i montažnih sustava.

| Zadatak zadan: | Rok predaje rada: | Predviđeni datumi obrane: |
|---|---|---|
| 06. prosinca 2012. | 07. veljače 2012. | 13.- 15. veljače 2012. |

Zadatak zadao: Predsjednik Povjerenstva:

Prof. dr. sc. Bojan Jerbić Prof. dr. sc. Franjo Cajner

# SADRŽAJ

# POPIS SLIKA

# POPIS TABLICA

# POPIS OZNAKA

| Oznaka | Jedinica | Opis |
|---|---|---|
| $L$ | [m] | Duljina |
| $f$ | [kHz] | Frekvencija |
| | [slika/sec] / FPS | Brzina izmjene slike |
| $t$ | [sec] | Vrijeme u sekundama |
| $m$ | [kg] / [g] | Masa |
| $v$ | [°/rad] | Brzina |
| $\alpha$ | [°] | Kut |
| | [mm] | Ponovljivost |

# SAŽETAK

Roboti u današnje vrijeme još uvijek egzistiraju kao zatvorene cjeline u industrijskoj primjeni zbog svoje potencijalne opasnosti za ljude. No, u laboratorijskim istraživanjima i uvjetima, moguće je zbližiti čovjeka i robota tako da oni dijele radni prostor. U radu je na svojevrstan način prikazano korištenje intuitivnog upravljanja robota uz pomoć Microsoftovog Kinect senzora kao ulazne jedinice za korisnika. Razvijena je aplikacija za interaktivno komuniciranje korisnika i robota na način da je robot u stanju ponoviti naučene radnje u raznim varijacijama kada mu korisnik to zada.

Ključne riječi: Kinect, robot, gesta, interakcija, robotski vizijski sustav

# SUMMARY

Robots in today industrial use still exist as closed entities due to their potential danger for people around them. While, in laboratory research where you can find laboratory conditions, it is possible to put human and robot in one cell, so that they share working space. In this thesis it is presented intuitive way of robot control using Microsoft Kinect sensor by means of input device for user. Application for interactive communication between user and robot is developed so that robot is capable to repeat learned actions in various variations when user gives him command.

Key words: Kinect, robot, gesture, interaction, robotic vision system

# 1 UVOD

Korištenje robota u vidu programabilnog manipulatora od svoje pojave pa do danas bazira se ponajviše na uporabi u industrijskoj proizvodnji gdje je zahtjev da se proizvodi efikasnije od čovjeka kao radne snage. Da bi roboti funkcionirali, moraju se na neki način naučiti da rade ono što se od njih očekuje, u najvećem broju slučaja programirati na određen način. U novije doba programiranje više nije tako rudimentarno i u robote se počinju ugrađivati i elementi inteligencije tako da im se prilikom programiranja ugrađuju algoritmi koji omogućavaju da robot primjerice sam spozna okolinu oko sebe uz pomoć senzora. Kroz proces spoznaje okoline, robot, kao i čovjek uči i spreman je uspješno obavljati specifične problema za koje je namijenjen. Programiranje je neophodan dio rada s robotima, ali i općenito svim modernijim strojevima, kako alatnim, tako i onima opće uporabe. No, moguće je izbjeći programiranje u slučajevima poput ovoga kada se primjerice od robota očekuju jednostavnije radnje koje je prije naučio ili njihova kombinacija. U tu se svrhu, primjerice, može koristiti stereo-vizijski sustav Kinect, što je i tema ovog rada. Nastala je ideja da obični korisnik koji nema iskustva ni potreba znati programirati robota, može obavljati jednostavnije radnje, na jednostavan, logičan i intuitivan način. Jedno takvo rješenje će biti prikazano u okviru ovog diplomskog rada.

# 2 Zadatak i plan rada

Zadatak rada je intuitivno upravljanje robotom prilikom rukovanja objektima.

Za postizanje rezultata planirano je u prvoj fazi istražiti mogućnosti Kinect senzora i njegovu točnost.

Aplikacija izrađena u ovom radu koristit će se za intuitivno upravljanje robota. Korisnikove će kretnje tako cijelo vrijeme biti praćene Kinect - om, a željena će se akcija odvijati nakon odabira određene radnje na interaktivnom sučelju, i to tako što će korisnik rukom označiti željenu radnju. Na taj način korisnik upravlja robotom bez poznavanja programiranja i samih procesa koji se odvijaju da bi se postigla tražena akcija, što olakšava upravljanje. U radu bi trebala biti podrška za par različitih predmeta rada, tako da se može prezentirati rad sustava, no poznavanjem vizijskih procesa moguće je lako dodati i druge predmete rada.

Aplikacija za upravljanje bi trebala biti jednostavnog izgleda i razumljiva korisniku koji se nikad prije nije susreo s njom, a način na koji će se to ostvariti biti će razrađen u radu. Programiranje je potrebno izvesti u C# programskom jeziku u Visual Studio 2012 okruženju. Unutar aplikacije bilo bi poželjno ugraditi i prepoznavanje geste korisnika.

Robot kojim se raspolaže je Fanuc M-3iA, sa fiksiranim vizijskim sustavom, no moguća je primjena na bilo kojem Fanuc-ovom robotu koji ima vizijski sustav i podršku za programski paket Roboguide i programski jezik karel u kojemu treba napisati kod za komunikaciju aplikacije i robota.

Prvo bi bilo potrebno odrediti koji će obim radnji robot obavljati pa je on sveden na „prihvati i odloži“ radnju s 3 predmeta rada, čisto za prezentaciju rada sustava.

Treća faza je izrada dijagrama toka i algoritma za aplikaciju kojom će se služiti krajnji korisnik.

Nakon temeljnog stvaranja aplikacije potrebno je u nju ugraditi dio koji će biti odgovoran za komunikaciju robota i aplikacije, u krajnjem slučaju – korisnika.

Nadalje, nakon testiranja komunikacije potrebno je naučiti robota radnje s objektima koristeći vizijski sustav koji je na raspolaganju.

Nakon toga potrebno je objediniti naučene radnje s aplikacijom i prilagoditi algoritam aplikacije njenom dijagramu toka te testirati sam rad.

U prvom dijelu rada će biti ukratko opisan Microsoft Kinect senzor, sa svojim mogućnostima i primjenom u robotici.

Nadalje će biti opisan heksapodni robot koji je korišten u diplomskom radu Fanuc M-3iA, sa svojim karakteristikama i mogućnostima, kao i oprema koja je još korištena u Laboratoriju za projektiranje izradbenih i montažnih sustava.

Nakon toga biti će detaljno obrazložena izrađena aplikacija za upravljanje s osvrtom na propuste i poboljšanja te daljnji razvoj.

U zadnjem dijelu je dan prikaz testiranja rada sustava i zaključak.

# 3 Microsoft Kinect

Kad se po prvi puta pojavio na tržištu, u studenom 2010. bio je hit za konzolu Microsoft Xbox, za koji je primarno razvijen. No, razvojem događaja i uviđanjem mogućnosti takvog uređaja, pojavila se ideja i potreba za omogućavanjem korištenja uređaja i na Windows operativnom sustavu, što nije bio problem, jer je i on u vlasništvu Microsofta, pa tako od 1. veljače 2012. Kinect se može koristi i na osobnim računalima uz upotrebu ne komercijalnog SDK (eng. *Softvare Developement Kit*). On danas svoju nalazi u raznim područjima kao što su grafička obrada, prezentiranje, obrazovanje, podrška u robotici, komercijalne primjene.

U vrijeme kad se pojavio, Kinect je bio popularan prvenstveno radi svoje cijene koja je oko 1000 HRK, u usporedbi s ostalim stereo-vizijskim sustavima poput Bumblebee Stereo Vision System čija cijena je oko 2000 USD, uz koji se razvojni softver još i dodatno plaća, pa je cijena još i veća. Naravno, mogućnosti i karakteristike se razlikuju, no Kinect je prvenstveno i namijenjen niskobudžetnim rješenjima kao što su konzole, no ima potencijal za širenje svojih mogućnosti korištenjem SDK.

SDK je besplatan softver koji se može dobiti na Microsoftovim stranicama a koji sadržava osnovne primjere i mogućnosti uređaja, kao i upute za korištenje. Brzorastući razvoj i sve više primjera primjene zajedno sa besplatnom podrškom kako službenom, tako i neslužbenom na forumima osiguravaju kompletnost proizvoda, tj. proizvod i podršku za njega, što je bitno za ovakav relativno novi proizvod.

## 3.1 Tehnički podaci – tehnologija Kinecta

Kinect je kompaktan elektronički uređaj izduženog oblika na pomičnom stalku.

Na slikama (slika 1 i slika 2) vidi se sam uređaj i oprema koja se dobije u paketu prilikom kupnje uređaja.


**Slika 1 Kinect [1]**

**Slika 2 Oprema [2]**

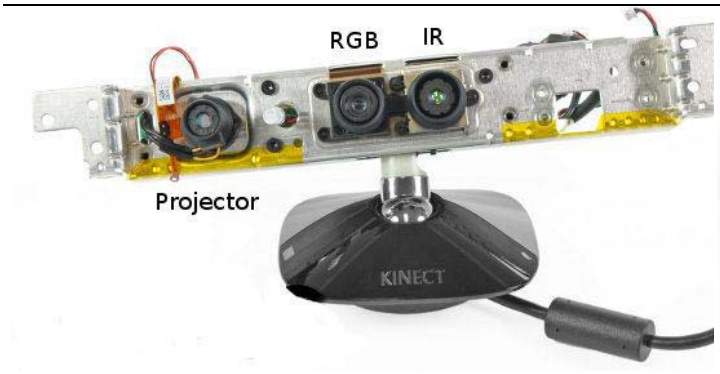
Na slici 1 se mogu vidjeti osnovni dijelovi uređaja, a to su sam uređaj sa senzorima i zvučnicima te motorizirano postolje.

Prikaz po komponentama se može vidjeti na slici ispod.


**Slika 3 Potpuno rastavljen uređaj [3]**

Budući da se na slici (slika 3) vide svi dijelovi koji sačinjavaju proizvod, potrebno je malo detaljnije obraditi bitnije komponente a to je dakako vizijski sustav.

„Oči“ Kinecta su 3 senzora (dvije kamere i jedan infra-crveni projektor), a mogu se pogledati na slikama (slika 4 i slika 5).




**Slika 4 Prikaz bez kućišta [4]**

**Slika 5 Kamere i senzor [3]**

Na slici 5 s lijeva na desno mogu se vidjeti:

- (Color CMOS) VNA38209015
- (IR CMOS) Microsoft / X853750001 / VCA379C7130
- (IR Projector) OG12 / 0956 / D306 / JG05A

### 3.1.1 Tehničke specifikacije

**Značajke senzora (uređaj)** [5]**:**

- Optički senzori (dvije kamere i jedan IR senzor)
- 4 mikrofona
- Motor za pomicanje
- Potpuna kompatibilnost sa XBOX 360 i Windows PC uređajima

**Raspon vizijskog sustava** [5]**:**

- Horizontalni raspon vidnog polja: 57 stupnjeva
- Vertikalni raspon vidnog polja: 43 stupnjeva
- Mogućnost pomicanja pomoću motoriziranog postolja : ± 28 stupnjeva po vertikali
- Raspon senzora dubine: 1.2 m - 3.5 m

**Protok podataka** [5]**:**

- 320x240 16-bit izlazna rezolucija podataka dubine pri 30 sličica/sek
- 640x480 32-bit izlazna rezolucija podataka boje pri 30 sličica/sek
- 16-bit audio @ 16 kHz

**Sustav praćenja skeleta** [5]**:**

- Prati do 6 osoba, uključujući 2 aktivna korisnika

- Prati 20 „zglobova“ po aktivnom korisniku
- Mogućnost povezivanja aktivnih igrača sa LIVE avatarima – vezano uz konzolu

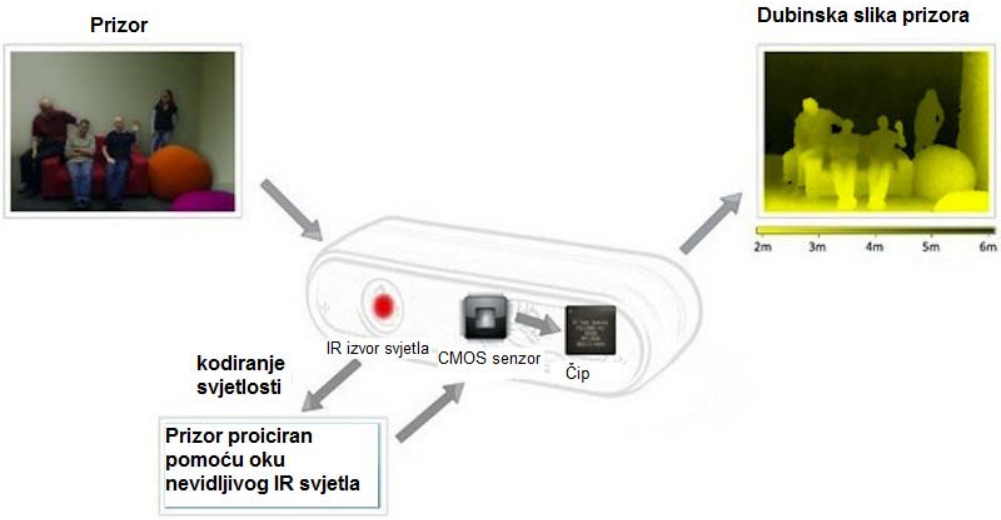
**Audio sustav** [5]**:**

- LIVE grupni razgovor i razgovor unutar igara
- Sustav poništavanje jeke koji naglašava govor
- Prepoznavanje govora podržanih jezika

## 3.2 Stereo-vizijski sustav

Sustav strojnog vida koji koristi minimalno dvije fizički odvojene kamere, ili jednu pomičnu, naziva se stereo-vizijski sustav. Općenita namjena mu je da omogući stroju da dobije podatke o dubini (udaljenosti) predmeta koji promatra. To se postiže upravo uporabom dvije kamere koje istovremeno promatraju neki objekt, svaka pod svojim kutom, pa se podaci iz obje uspoređuju i na temelju razlika piksela može se izračunati udaljenost objekta (skupine piksela) od kamere.

Na slici (slika 6) je prikazana shema stereo-vizijskog sustava koji koristi Kinect, a on je razvijen od strane izraelske tvrtke *PrimeSense*.

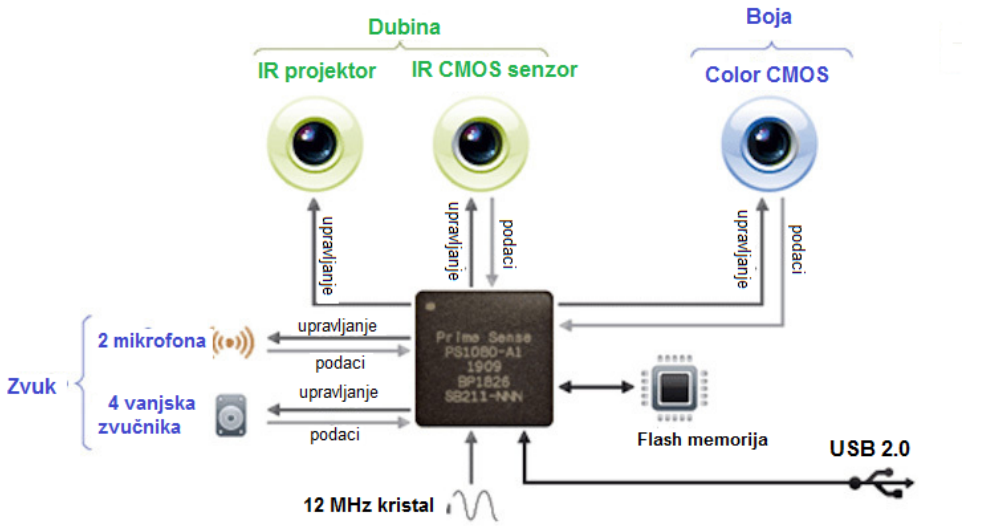



**Slika 6 Prikaz dobivanja slike i dubine [6]**

PrimeSense 3D senzor vidi i prati kretnje korisnika u okolini i pruža aplikacijama upravljačke signale. Na taj način je omogućen jednostavan API koji prevodi pokrete u determinirane ulaze kod aplikacije. Cjelokupna aktivnost se izvršava bez ikakvih pretpostavki o aplikaciji koja će ga koristiti, korisniku ili njegovoj okolini. Ne koristi se potrošna roba, pa je senzor praktični

jednostavan za korištenje. Cijeli senzor je temeljen na čipu PS 1080 Soc. Na slici (slika 7) se vidi pojednostavljena shema iskorištenja čipa. Čip kontrolira IR izvor svjetlosti (projektor), kako bi se projicirala slika i preobličila u IR kodiranu sliku. Za to se koristi standardni CMOS senzor, koji prima projicirano IR svjetlo i prebacuje IR kodiranu sliku do samog čipa. Čip procesira IR sliku i stvara točnu dubinu slike u sličica/sek.

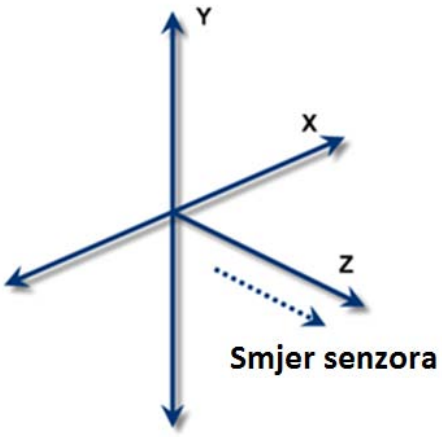
Rad samog čipa prikazan je na sljedećoj slici (slika 7).




**Slika 7 Shema rada PrimeSense čipa [7]**

Čip PS1080 SoC podržava više senzora, i tako omogućava sinkronizaciju dubine slike, boju te primanje zvuka. Čip koristi USB 2.0 sučelje za komunikaciju i prijenos podataka aplikaciji koja ga koristi. Budući da je čip građen tako da nema nikakvih pretpostavki o aplikaciji ili daljnjem procesoru koji će ga koristi, on je potpuno otvoren za svaku smislenu primjenu. Svi algoritmi za primanje i procesiranje se odvijaju na samom čipu, dok se samo minimalni komunikacijski sloj koji je potreban za USB sučelje odvija na strani aplikacije/procesora. To omogućava dobivanje podataka o dubini čak i kod uređaja i primjena koje imaju ograničene resurse procesora.

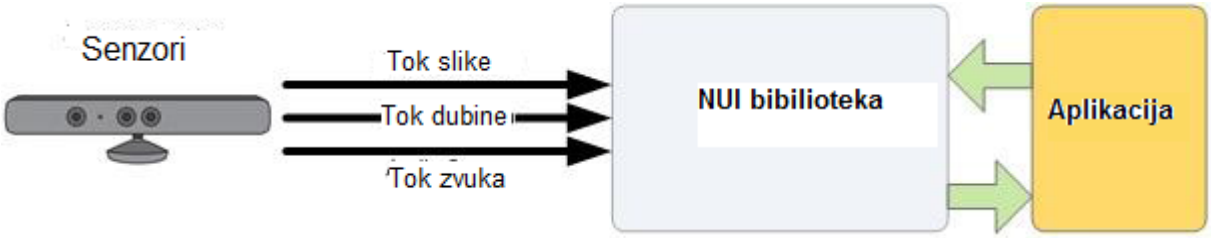
Koordinatni sustav Kinecta je desnokretni, i postavljen je kako je prikazano slikom (slika 8).




**Slika 8 Koordinatni sustav Kinect senzora [8]**

### 3.2.1 Arhitektura Kinecta namijenjenog za Windows

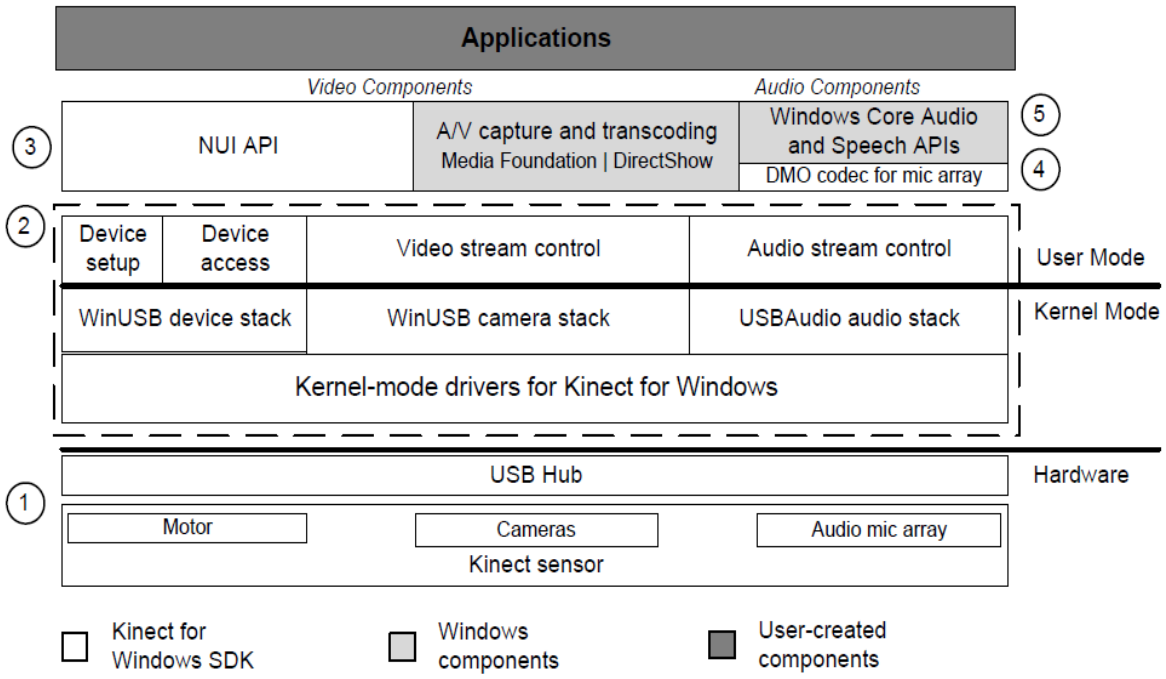
Prilikom korištenja Kinecta na Windows operativnom sustavu, potreban je Beta SDK, što je paket koji sadrži softverske i hardverske komponente (eng. *libraries and tools*). Sam Kinect senzor skupa sa pripadajućim bibliotekom koja sadrži upute za NUI radi na način prikazan slikom (slika 9).




**Slika 9 Poveznica softvera i hardvera s aplikacijom [9]**

Za razliku od slike 7, na slici 9 su pojednostavljeno prikazani tokovi, koji se odvijaju za vrijeme korištenja uređaja.

Nadalje, prikazana je sama arhitektura SDK.




**Slika 10 Beta SDK arhitektura [9]**

Komponente koje SDK sadrži:

1. Kinect hardver – u njih su uključeni Kinect senzor i USB hub, kojim je senzor spojen sa računalom(aplikacijom)
2. Microsoft Kinect drajveri

   Windows 7 drajveri za kinect senzor koji su instalirani u sklopu SDK:

- Kinect senzor mrežu mikrofona kao kernel mod audio uređaj kojem je moguće pristupiti koristeći standardne audio API (eng. *application programming interface*) unutar Windowsa.
- Tok podataka slike i dubine.
- Mogućnost korištenja više od jednog Kinect-a na jednom uređaju

3. NUI API – set API-a za očitanje podataka sa senzora.
4. Kinect Audio DMO

   Kinect-ov DMO (eng. *DirectX Media Object*) povećava mogućnosti Windows 7 te koristi funkcije za usmjereno snimanje zvuka (eng. *beamforming)* i lokalizaciju izvora zvuka (eng. *source localization).*

Windows 7 standardni API, uključuje audio, multimediju i govor unutar Windows 7 i Microsoft Speech SDK.

## 3.3 NUI

Već je spomenuto da je Kinect senzor za praćenje pokreta, što ga svrstava su uređaje koji imaju NUI.

Natural User Interface je novije sučelje između korisnika i stroja (u ovom slučaju konzole ili računala/uređaja izvršitelja), za razliku od starijih sučelja kao što su CLI i GUI.

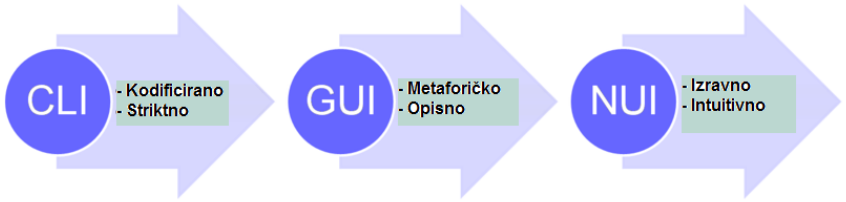
Korisničko sučelje može se definirati kao veza između korisnika i uređaja kojim upravlja tako je najprije već u 1970 ima osmišljen CLI (eng. *Command-Line Interface*) koji je bio direktna veza sa tadašnjim računalom. Komande su bile kodiranje i striktne, te je bilo potrebno točno znati strukturu računala da bi se moglo njime nešto načiniti.

GUI (eng. *Graphical User Interface*) je začet već u 1980-ima na Xerox računalima, da bi svoju ekspanziju započelo pojavom Windowsa. Ovdje je riječ o sučelju gdje korisnik interakcijom sa grafičkim entitetima daje naredbe uređaju, dok se naredbe odvijaju u pozadini, i ne opterećuju korisnika. Danas je najraširenije sučelje za upravljanje elektroničkim uređajima, pa se tako koristi i kod strojeva, mobitela, u komercijalne svrhe itd.

Konačno, u sadašnje vrijeme sve prisutnije jr sučelje NUI (eng. *Natural User Interface*), koje ima mnogo svojih inačica, a može se shvatiti kao i virtualna stvarnost. Važno je napomenuti da su i prijašnja 2 sučelja nazivana NUI, jer u doba svojeg nastanka su to i bili. Današnji NUI ne može bez grafičke reprezentacije, a ima, naravno i komande koje se izvršavaju u pozadini. Stoga se može reći da on objedinjuje prijašnja dva sučelja, te dodaje nov način interakcije korisnika sa uređajem, koji se može reći da je prirodan, Takav način rada možemo nazvati prirodnim, čemu svjedoče geste i govor, karakteristične i normalne ljudima, a to su geste i govor, što i jest karakteristično i prirodno za ljude. No upotreba takvog sučelja nije još uzela svoje veliki zamah, i zato je tu veliki potencijal za nova rješenja i olakšavanje upotrebe korištenja takvih uređaja, te naravno smanjenje cijene. Trenutna mana takvog sučelja je naravno programiranje senzora i uređaja koji bi interpretirali korisnikove želje, a to se u današnje vrijeme pokušava riješiti primjenom kognitivnog pristupa, o kojem je već bilo riječi u prethodnom poglavlju.

Baš zbog svoje inovativnosti i intuitivnosti, smatram da je Kinect zapravo sučelje između korisnika i uređaja kojim se upravlja, bilo to igraća konzola, računalo, robot, stroj ili nešto sasvim drugo, naglasak je uvijek na tome je riječ o sučelju

Na slici (slika 11) je grafički predočen razvoj korisničkih sučelja.




**Slika 11 Razvoj korisničkih sučelja [10]**

## 3.4 NUI API [11]

NUI API je sama Kinect za Windows API. Podržava temeljne upravljačke značajke za procesiranje slike i uređaja:

- Pristup Kinect senzorima koji su spojeni na računalo.
- Pristup na podatke o toku (eng. *stream*) slike i dubine iz Kinect senzora.
- Prikazuje obrađenu verziju senzora dubine i slike, kako bi podržao praćenje pokreta kroz praćenje skeleta (eng. *skeletal tracking).*

## 3.5 NUI tok podataka RGB slike

Podaci se isporučuju kao niz slika na kojima se objekti ne miču, a zbog njihove brze izmjene mozak to registrira kao pokret. Nakon NUI inicijalizacije, program identificira tokove koje planira koristiti. Nakon toga otvara one tokove s dodatnim specifičnim detaljima, uključujući i podatke o razlučivosti, vrsti slike, broj među spremnika, te koji *runtime*[1]je potrebno koristiti za pohranu dolaznih slika (eng. *frame*). Ako *runtime* ispuni sve međuspremnike prije nego aplikacija dobije zahtijevani tok, *runtime* odbacuje najstariju sliku i ponovno koristi međuspremnik. Kao rezultat toga, moguće je da određeni broj slika bude odbačen tj. neiskorišten. Aplikacija može zatražiti do četiri među spremnika, ali za većinu scenarija korištenja dovoljno je dva među spremnika.

Aplikacija ima pristup sljedećim vrstama slikovnih podataka pristiglih iz senzora:

1. Podatci o boji
2. Podatci o dubini
3. Podatci o segmentaciji korisnika

Nadalje će biti pojašnjen svaki od skupova podataka.

[1] eng. *runtime* predstavlja vrijeme tijekom kojega se program izvršava, za razliku od primjerice od drugih različitih vremena kao što je *compile time.*

### 3.5.1 Podatci o slici

Podatci koji se odnose na RGB sliku dostupni su u sljedeća dva formata:

RGB boja daje 32-bitni, linearni X8R8G8B8 formatiranu bitmapu u boji ( eng. *formatted color bitmap),* u sRGB prostor boja. Za rad s RGB podataka, aplikacija bi trebala odrediti boju ili color_YUV vrstu slike kad se otvori tok podataka.

YUV boja daje 16-bitni, gama-korektiran (eng. *gamma-corrected)* linearni UYVY-formatiranu bitmapu u boji (eng. *formatted color bitmap),* gdje je gama korekcija u YUV prostor jednaka sRGB gama u RGB prostoru. Budući da YUV protok koristi 16 bita po pikselu, ovaj format koristi manje memorije i dodjeljuje manji među spremnik kada je *stream*[2]otvoren. Za rad s YUV podatcima, zahtjev treba navesti „raw" YUV vrstu slike kad se otvori potok. YUV podatci su dostupna samo na 640x480 rezoluciji i samo na 15 FPS.

Oba formata se izračunavaju iz istih podataka kamere, tako da YUV podatci i RGB podatci predstavljaju istu sliku. Potrebno je odabrati tip podatka koji naviše odgovara određenoj aplikaciji i njezinoj primjeni.

Senzori koriste USB vezu za prijenos podataka na računalo, ali ne smije se zaboraviti da USB veza ima svoj maksimum protočnosti koji nije zanemariv. Podatci o slici koje senzor prikazuje na 1280x1024 se komprimiraju i konvertiraju u RGB prije prenošenja podataka u *runtime*. *Runtime* zatim dekomprimira podatke prije nego što ih proslijedi do aplikacije. Korištenje kompresije omogućuje prikaz podataka o boji u 30 FPS, ali algoritam koji se koristi dovodi do gubitka nekih detalja slike.

### 3.5.2 Podatci o dubini

Tok podataka dubine (eng. *deapth stream*) prikazuje slike u kojima svaki piksel predstavlja udaljenost u kartezijskom koordinatnom sustavu u milimetrima od kamere do najbližeg objekta određene x i y koordinate u polju vidljivosti senzora. Na raspolaganju su slijedeći tokovi podataka:

- Veličina prikaza od 640×480 piksela
- Veličina prikaza od 320×240 piksela
- Veličina prikaza od 80×60 piksela

[2]eng. *stream* u ovom slučaju predstavlja tok podataka

Aplikacije mogu procesuirati podatke o dubini kako bi omogućile upotrebu raznih značajki kao što su primjerice praćenje korisnika ili identifikacija pozadinskih objekta, a u svrhu njihova ignoriranja.

Format podatka o dubini ovisi o tome specificira li aplikacija samo dubini ili i dubinu i indeks korisnika kod pokretanja NUI-a:

- Ako je samo dubina onda 12 bitova zauzima podatke o dubina a preostala 4 su neiskorištena.
- Ako je dubina i korisnikov indeks onda prva 4 bita zauzimaju podatke o indeksu a ostatak o dubini.

Vrijednost dubine 0 označava da nije dostupan podatak o dubini, u slučaju kada su objekti preblizu ili predaleko kameri.

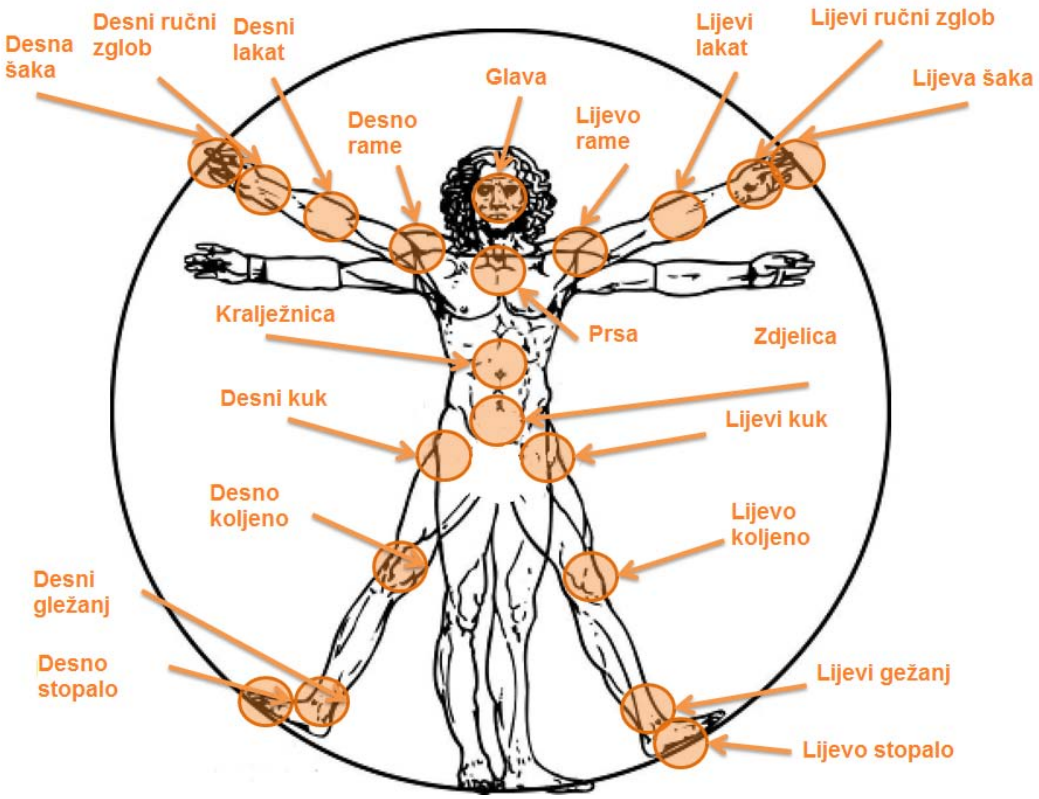
### 3.5.3 Podatci o segmentaciji korisnika

U beta SDK-u, Kinect procesira podatke iz senzora kako bi identificirao 2 ljudske figure ispred Kinect-a, a potom kreira mapu segmentiranih korisnika (eng. P*layer segmentation mapu)*. Ova mapa je *bitmap* u kojem svaka vrijednost piksela ima određenu vrijednost korisnika (igrača, objekta promatranja) i to onu kojem se nalazi bliže, a to vrijedi za svaki piksel. Iako je segmentacija podataka korisnika odvojeni logički sistem u praksi podatci o dubini i podatci o korisniku su sjedinjeni u jedan sliku.

Korisnikov indeks može biti i nula, a to onda znači da nije pronađen niti jedan korisnik. Aplikacije uobičajeno koriste korisnikove podatke kao masku za izoliranje specifičnih korisnika ili objekata interesa od sirovih podataka boje i dubine.

### 3.5.4 Praćenje skeleta

NUI *skeleton* API daje informacije o lokaciji 2 korisnika uključujući i njihovu orijentaciju. Podatci su u obliku anatomskih značajki na ljudskom tijelu mahom zglobova (eng. *joints*) prikazani na slici ispod (slika 12). To je ukupno 20 značajki na ljudskom tijelu koje su sadržani unutar Microsoft.Kinect.dll biblioteka (eng. *library*) datoteke koja se koristi kod razvoja aplikacija za Kinect, a koja je korištena i u radu.



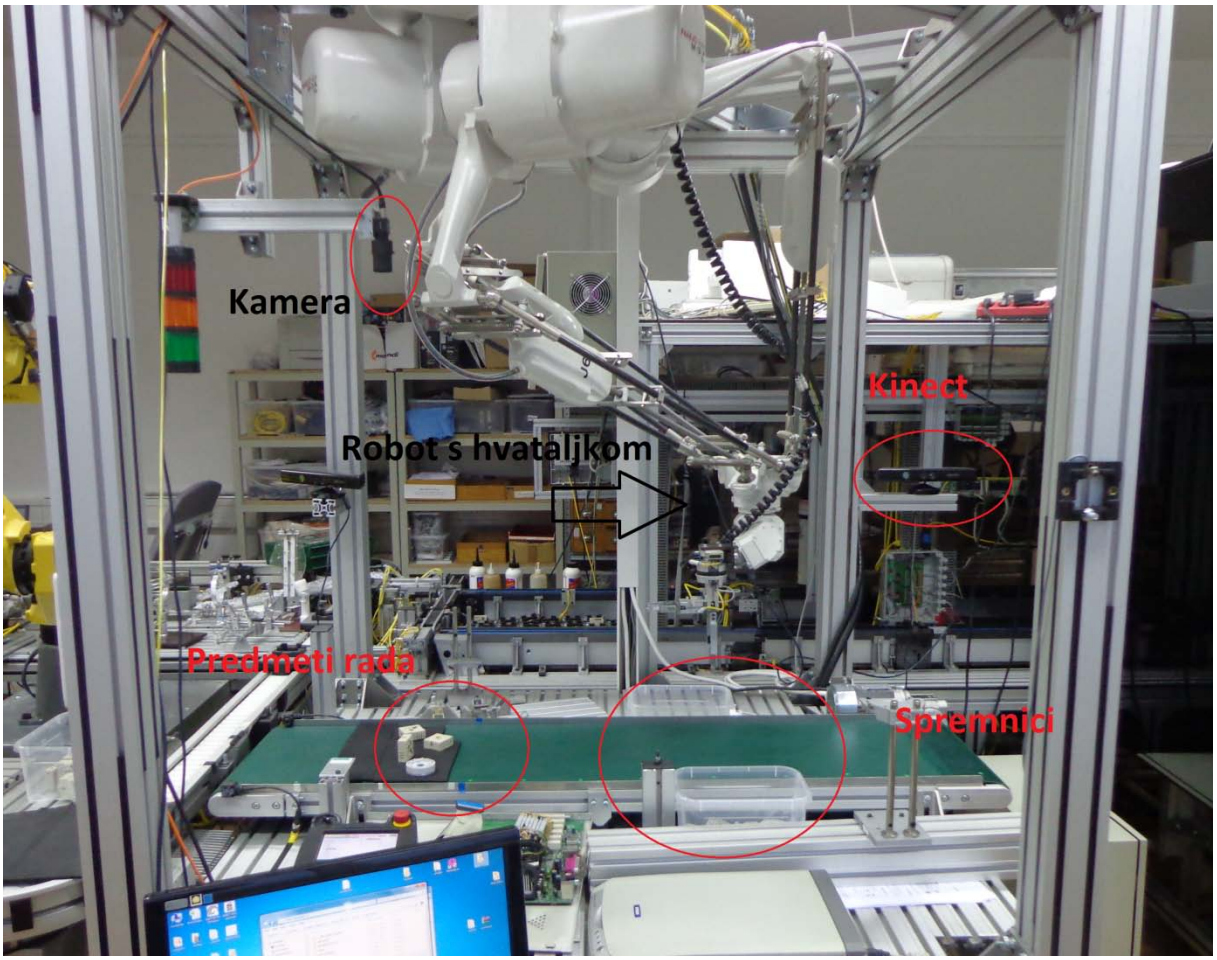

**Slika 12 Prikaz značajki koje Kinect može pratiti [8]**

# 4 Oprema korištena u radu

Praktični dio rada je najvećim dijelom izveden u laboratoriju za projektiranje automatskih montažnih sustava. Korištena je sljedeća oprema:

1. Microsoft Kinect senzor
2. Heksapodni robot Fanuc M-3iA s pripadajućom upravljačkom jedinicom
3. Jednostavna pneumatska hvataljka za robota
4. Vizijski sustav kojeg sačinjava kamara Sony XC-56
5. Osobno računalo
6. Radni stol i odgovarajuća robotska stanica za visećeg robota

Na slici (slika 13) se može vidjeti prostorni raspored robotske ćelije gdje je izvršen rad i testiranje sustava.




**Slika 13 Oprema u laboratoriju - dijelovi sustava**

Kao što se može vidjeti na slici 13, prostorni je raspored elemenata sustava smješten unutar robotske ćelije s visećom instalacijom robota. Predmeti rada se nalaze na pokretnoj traci koja je fiksirana, jer nama potrebe niti mogućnosti za potrebe rada. Kinect je smješten na posebnom nosaču, i njegova pozicija može varirati. Bitno je samo da može uspješno pratiti korisnika na udaljenosti od 1 do 3 m, ta da u tom području korisnik može istodobno pratiti gibanje robota i stanje aplikacije na monitoru. Kamera je također pričvršćena na posebnom nosaču i usmjerena je na određeni prostor na traci gdje može snimati radne predmete. Upravljačka jedinica robota se nalazi ispod stola. U daljnjem dijelu poglavlja bit će detaljnije opisana pojedina spomenuta oprema.

Treba istaknuti da je oprema preuzeta u ispravnom stanju i da je takva i razdužena.

## 4.1 Heksapodni robot FANUC M-3iA

Heksapodni roboti su uvelike različiti po svojoj kinematici i dinamici od standardnih, mahom robota s rotacijskom strukturom u obliku robotske ruka sa rotacijskim zglobovima. Nadalje, drugačiji je i raspored masa, čime se dolazi i do različitih inercija, pa tako primjerice heksapodni robot može brže izvoditi operacije s predmetima, jer je veličina njegove mase smještena u bazi robota, dok su samo manji dio i sam alat smješten na kraju robota.

Mana robota je smanjen obujam radnog prostora u odnosu na robot s rukom. Fotografija robota korištenog u radu je na slici (slika 14).

**Slika 14 Fanuc M-3iA [12]**

## 4.2 Karakteristike robota

Robot je namijenjen za operacije s manjim predmetima ali pri velikim brzinama. Ima veliku fleksibilnost jer je oblik krajnjeg zgloba takva da je moguća ugradnja alata s primjerice dvije hvataljke koji se izmjenjuju okretom predzadnjeg zgloba.

**Tablica 1 Karakteristike robota [12]**

| Karakteristika | Mjerljiva veličina |
|---|---|
| **Kontroler** | R-30iA Mate |
| **Broj upravljanih osi** | 6 |
| **Maksimalni kapacitet [kg]** | 6 |
| **Ponovljivost [mm]** | ± 0.1 |
| **Masa robota [kg]** | 153 |
| **Dimenzije radnog prostora [mm]** | 1350 x 500 |
| **Maksimalna brzina za zadnja 3 zgloba [°/s]** | 2000 |

U tablici 1, kad se spominje 6 upravljanih osi, misli se na 3 osi (X,Y,Z) i 3 rotacije oko tih osi, ili neke druge referentne geometrijske veličine. Tri rotacije se mogu shvatiti kao rotacija, precesija i nutacija.

Upotreba robota je idealna za prehrambenu industriju, jer može izvršavati radnju preslagivanja, umetanja ili izuzimanja predmeta iz ambalaže velikom brzinom, jer su prehrambeni proizvodi većinom malih dimenzija i puno ih ima.

Upravljačka jedinica (kontroler) robota R-30iA Mate je dosta kompaktna i mala u odnosu na ostale za slične robote. Svrha upravljačke jedinice je da sve naredbe robotu dane robotu od strane korisnika prevede u strojni jezik. Također upravlja svim sustavima robota, kao što su hidraulički, mjerni, pneumatski. Upravljačka jedinica se vidi na slici (slika 15).

**Slika 15 Upravljačka jedinica R-30iA Mate [12]**

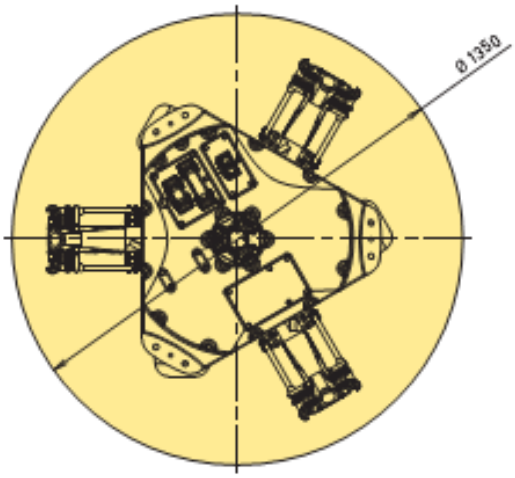
**Slika 16 *i*Pendant [12]**

Korisnik robotom može upravljati koristeći se privjeskom za učenje (*teach pendant*). Može se vidjeti na slici (slika 16). Ovakav način upravljanje je „on line“ što znači da robot odmah izvodi radnje koje mu se zadaju.

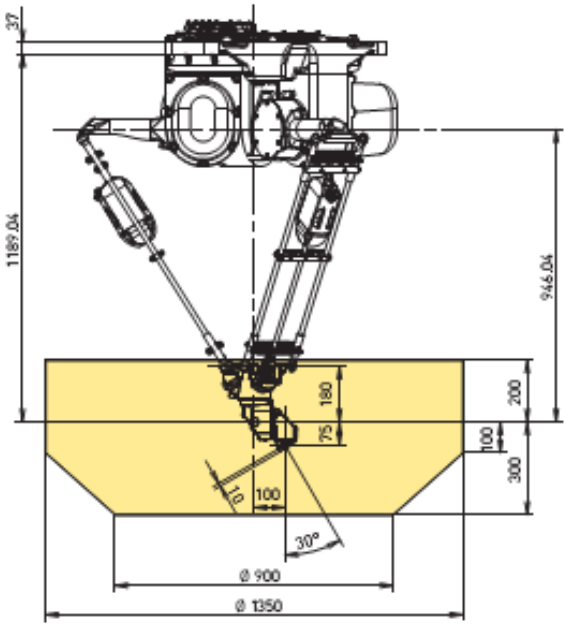
Privjesak za učenje ima nekoliko bitnih značajki koje su karakteristične svim privjescima, neovisno o tipu i proizvođaču industrijskog robota. To su sigurnosne značajke, prvenstveno za korisnika, ali iza robota. Naime, da bi robot mogao izvršiti zadano gibanje korisnik mora držati stisnutu tipku „*deadman switch*“ koja se nalazi s donje strane privjeska. Dodatno, korisnik mora držati i tipku SHIFT, i tek tada robot ima mogućnost gibanja.

### 4.2.1 Radni prostor

Heksapodni roboti imaju relativno mali radni prostor u odnosu na primjerice robote s rotacijskom strukturom. Zbog svoje kinematike, koja omogućuje gibanje unutar prikazanih gabarita radnog prostora prikazanih slikama (slika 17 i slika 18), robot ima radni prostor oblika cilindra




**Slika 17 Radni prostor – tlocrt [12]**




**Slika 18 Radni prostor – nacrt [12]**

## 4.3 Vizijski sustav robota

Robot ima fiksiranu kameru koja je smještena iznad područja koje je predviđeno za radne predmete. Samim time kamera nije dio robota, nego dio cijelog sustava. Nedostatak fiksirane kamere je to što kad jednom snimi i locira predmet rada, robot kreće uzimati predmet, no ukoliko dođe do poremećaja s premetom rada, robot to više neće moći registrirati i ići će uzeti predmet „na slijepo“. Naravno, moguća je ugradnja kamere izravni na glavu robota tako da bute uz hvataljku ili alat, no za potrebe rada to nije bilo potrebno. Ugradnjom kamare bi se također smanjila dodatna nosivost predmeta rada, jer je nosivost glave ograničena.

### 4.3.1 Kamera korištena za vizijski proces

Fiksirana kamera vizijskog sustava je od strane Japanskog proizvođača Sony. To je kamera za crno-bijelo snimanje za industrijsku primjenu tip Sony XC-56. Kamera je osposobljena za industrijske uvjete i izdržljiva je u pogledu temperature i mehaničkih poremećaja. To se može vidjeti na slici (slika 19).


**Slika 19 Sony kamera XC-56 [13]**

Karakteristike kamere prikazane su tablicom (tablica 2).

**Tablica 2 Tehničke karakteristike kamere Sony XC-56 [13]**

| | |
|---|---|
| **Senzor slike** | Sony, 1/3 type Progressive Scan CCD |
| **Rezolucija** | VGA - 659 x 494 piksela |
| **Brzina okidanje** | 30 [FPS] (slika u sekundi) |
| **Prikaz slike** | Crno bijeli |
| **Priključci** | 12 pin |
| **Povećanje** | Fiksno/ručno |
| **Dimenzije (DxŠxV)** | 29 x 29 x 30 [mm] |
| **Namještanje leće** | C |
| **Napon** | DC 12 V (+10.5 to 15 V) |
| **Snaga** | 1.5 [W] |
| **Masa** | 150 [g] |
| **Radna temperatura** | -5 do 40 [°C] |

Da bi se na ispravan način moglo raditi u industrijskoj okolini, uz kameru je potreban i kvalitetan objektiv koji omogućava zadovoljavajući sliku za obradu. U konkretnom slučaju na kameri se koristi objektiv japanske tvrtke Tamron, model 12VM1040ASIR.
Karakteristike objektiva prikazane su tablicom (tablica 3).

**Tablica 3 Tehničke karakteristike objektiva [14]**

| | |
|---|---|
| **Žarišna duljina** | |
| **Fokus** | 0.5 - ∞ [m] |
| **Namještanje leće** | C |
| **Masa** | 77 [g] |
| **Radna temperatura** | -20 - +60 [°C] |

Objektiv je dovoljno dobar za snimanje statičkih predmeta, crno bijela slika također je dostatna za potrebe rada, čak je i bolje da je crno bijela, jer sustav radi brže i bolje može označiti rubove. Na slici (slika 20) se može vidjeti objektiv, koji se montira na spomenutu kameru u C načinu montaže.


**Slika 20 Objektiv Tamron 12VM1040ASIR [14]**

# 5 Aplikacija za upravljanje robotom

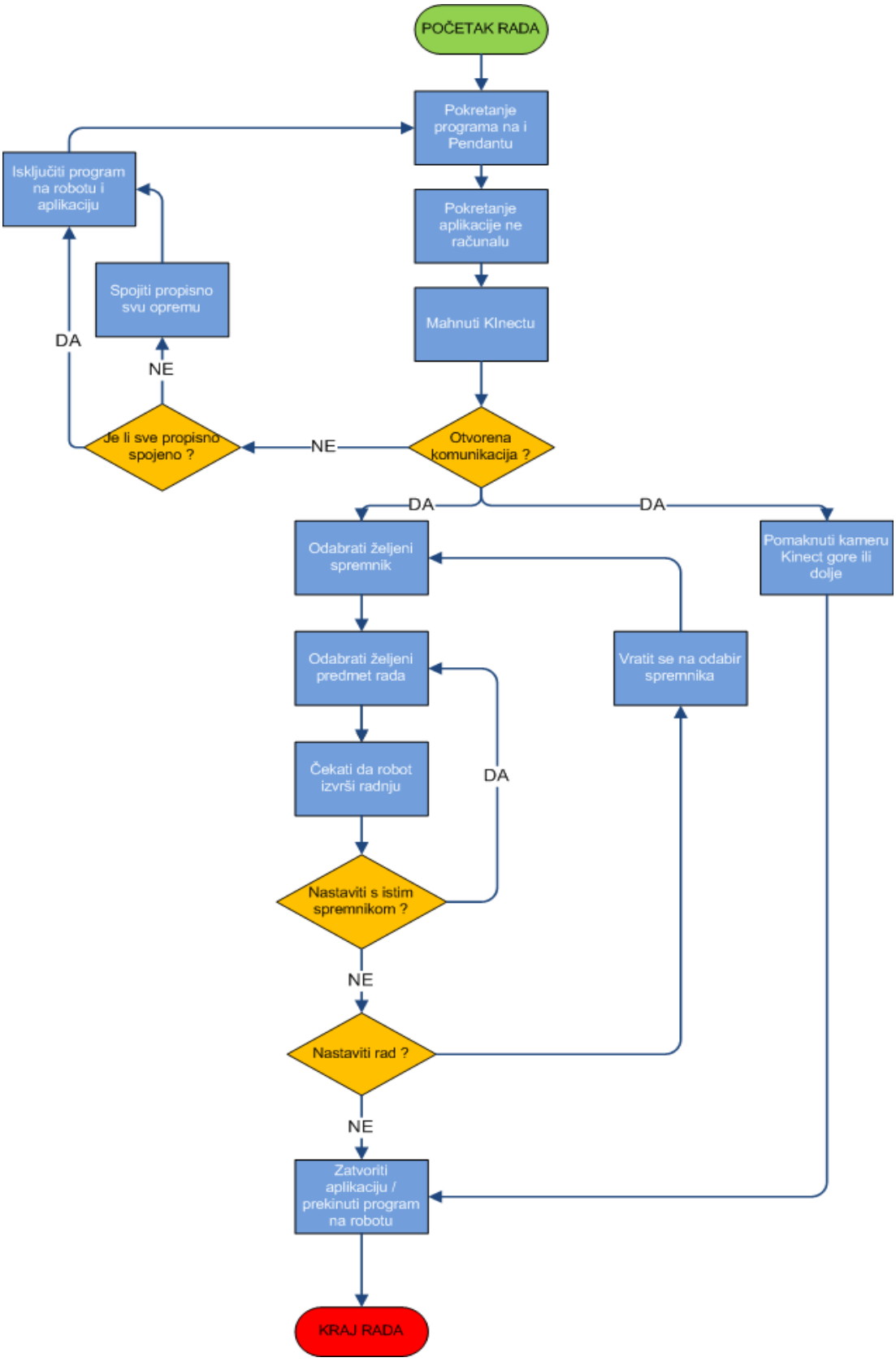
Aplikacija je napravljena kao interaktivna površina u prozoru, u čijoj pozadini je video koji snima Kinect senzor, a unutar prozora su interaktivni gumbi koji reagiraju kada ih korisnik pokrene. Prvotno je ideja bila koristiti što više gesti od strane korisnika kako bi se što više koristio „govor tijela“ korisnika za upravljanje robotom. No, ispitivanjem mogućnosti i pouzdanosti Kinect senzora, takvo upravljanje je svedeno na minimum. Cjelokupna aplikacija je programirana koristeći Microsoft Visual Studio 2012, u programskom jeziku C#, a bitno je za napomenuti da je za pravilan rad potreban Microsoft Network Framevork 4.5. Za rad aplikacije također je potreban Microsoft Developer Kit za Kinect, koji sadrži *Microsoft Kinect library* naredbi verzija 1.6 pošto je u toj verziji kod programiran, a kompatibilnost za buduću nadogradnju treba provjeriti kad izađe novija verzija.

Aplikacijski kod se sastoji od pozadinskog koda i koda za vizualni izgled. Pozadinski kod je čisti C# programski jezik, dok je vizualni izgled kodiran u xhtml jeziku. Oba koda u skraćenom obliku su priložena na kraju rada.

Na slici (slika XX) blok dijagramom je prikazan red sustava, od strane korisnika. Prikazuje koje su radnje koje korisnik može raditi pomoću sustava i može poslužiti kao orijentacijska pomoć za nekoga tko se nije susreo sa sličnim sustavom. Može se vidjeti da je rad sa sustavom baziran po određenim obrascima kako ne bi došlo do zabune korisnika, ili eventualnih šteta po sustav. Moguće su dakako mnogostruke nadogradnje sustava kao što je glasovno upravljanje, ili implementacija glasovnih naredbi u jednom dijelu upravljanja robotom.

Pokretanje aplikacije je moguće na 2 načina. Prvi način je jednostavno pokrenuti .exe datoteku u mapi „Release“, prilikom čega se otvara glavni prozor i aplikacija je spremna za korištenje.

Drugi način je napredniji i podrazumijeva da korisnik zna što želi napraviti. Prvo je potrebno u glavnom mapi otvoriti projekt same aplikacije koji ima ekstenziju .csproj, ili otvoriti rješenje (engl. *Solution*) projekta koje ima ekstenziju .sln. Nakon toga se otvara Visual Studio, i unutar njega dalje treba kompajlirati projekt, a potom ga pokrenuti pritiskom na tipku start, nakon čega će se otvoriti glavni prozor aplikacije. Drugi način je sigurniji, jer primjerice prilikom mijenjanja računala na koje se radi, .exe file neće raditi ako se promijene putovi do referentnih *library*-a koje koristi aplikacije a to su Microsoft Kinect i Coding4Fun.Kinect.Wpf.




**Slika 21 Pojednostavljeni algoritam rada sustava**

## 5.1 Poveznica s Kinect-om

Kako je spomenuto u poglavlju 3 (3.5), Kinect ima 3 toka podataka. U radu će se koristiti sva 3 toka, s time da će krajnjem korisniku biti vidljiv samo tok RGB slike prilikom rada aplikacije. Slika dubine i skeleta se izvode, i Kinect ih snima, a aplikacija obrađuje, no nisu potrebni za prikaz, i samo bi smetali u prozoru.

Kinect je u mogućnosti za ovu aplikaciju detektirati dvije osobe. Nadalje, od prepoznatih osoba ili jedne osobe, prepoznaje lijevu šaku (eng. *Left Hand*) ili desnu šaku (eng. *Right Hand*). To ovisi o tome koja je šaka najbliže samom senzoru. Naravno, ukoliko dođe to promjene udaljenosti šake korisnika, automatizmom se dalje prati opet najbliža šaka.

Praćenje šake se izvodi vizualnim elementom koji simbolički označava šaku, i pomiče se po prozoru kao kursor miša. Na slikama (slika 22 i slika 23) su prikazani vizualni elementi lijeve i desne šake koji se ponašaju kao kursor miša.

**Slika 22 Lijevi vizualni element**

**Slika 23 Desni vizualni element**

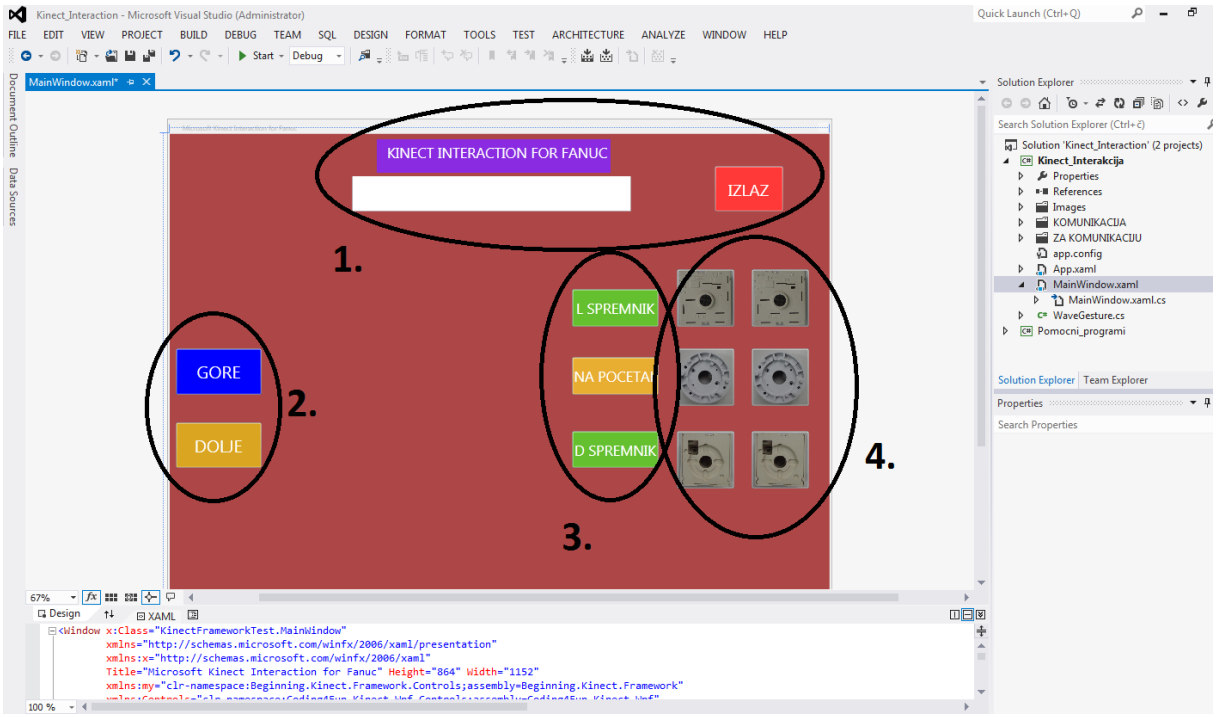
Odlika primjene Kinecta u ovom radu je to da je izbjegnuta potreba kalibriranja Kinecta. Naime, Kinect služi kao ulazna jedinica koja robotu ne šalje nikakve prostorne koordinate, niti ih ne prima, nego se sve trigonometrijske radnje obavljaju u aplikaciji pa se time pojednostavljuje cjelokupni rad sustava.

Prikaz videa s Kinecta je RGB prikaz, no slika koja se vidi na ekranu je zrcalna u odnosu na onu koju vidi sam Kinect. To bi možda moglo zbuniti korisnika koji podrobnije gleda sam video, no potpuno je svejedno je li slika zrcalna ili nije. Naime, pošto se i s tim prikazom dobiva efekt da kada korisnik pomakne ruku lijevo, i na ekranu se pomiče lijevo, korisnik se i ne koncentrira na to je li slika zrcalna ili nije, nego na praćenje svoje ruke, tj. kursora.

## 5.2 Glavni prozor

Glavni prozor, koji se otvara prije pokretanja aplikacije, ugrubo, sadrži 4 područja.

Na slici (slika 24) se jasno mogu i zamijetiti. Slika prikazuje isječak iz radne okoline Visual Studija, tj. prije samog pokretanja aplikacije.




**Slika 24 Aplikacija za korisnika**

Prvo područje se nalazi u gornjem dijelu i sačinjeno je od naslovnice i dinamične trake koja obavještava korisnika o tome kako koristiti aplikaciju i što očekivati od robota. Sadrži gumb za izlaz iz aplikacije, čijom aktivacijom se zatvara prozor i prekida veza s robotom.

Drugo područje je s lijeve strane i ima dva interaktivna gumba, koji služe za pomicanje Kinecta gore ili dolje. U kodu je namješteno da se Kinect pomiče prilikom aktiviranja gumba „GORE“ ili „DOLJE“ za + ili - 5 stupnjeva. Tako da u konačnici nije moguće postići krajnji doseg od -28 do + 28 stupnjeva. Valja napomenuti da nije preporučljivo stalno korištenje ove akcije pomicanja senzora, pošto motor koji pokreće Kinect nije predviđen za dugotrajni rad, i mogao bi se preopteretiti.

Sljedeća područja su skrivena prilikom pokretanja aplikacije a prikazuju se prilikom aktiviranja veze se robotom.

Treće područje je u sredini prozora i ono predstavlja glavni izbornik. Ovdje se mogu birati 2 spremnika u koje će se odlagati predmeti rada. Spremnici su smješteni s lijeve i desne strane trake na kojoj se nalaze predmeti rada.

Četvrto područje je rezervirano za predmete rada. Ono je također skriveno prilikom pokretanja aplikacije. Postaje vidljivo kada se odabere spremnik u 3. području. Mogu se

odabrati 3 predmeta rada. Kada se odabere željeni predmet rada, pokreće se robot koji počinje izvoditi radnju s predmetom, a u 3. području se aktivira gumb, koji služi za povratak na glavni izbor, a to je izbor spremnika.

Valja napomenuti da sučelje kako je prikazano na slici (slika 24) korisnik nikad neće vidjeti jer će u pozadini biti video s Kinecta, a ni svi gumbi neće biti aktivirani uvijek istovremeno.

Razine (engl *leayer*) u prozoru su također podijeljeni u skupine. Razine se koriste u kodiranju vizualnog izgleda, kako bi se raščlanile njihove uloge i omogućilo jednostavno prebacivanje iz razine u razinu, te grupiranje vizualnih elemenata pod određenom razinom.

Prva razina je osnovna razina prozora (eng. *window*), koja je neophodna za svaku WPF (engl. *Windows Presentation Foundation*) aplikaciju u Windows sustavu. Unutar prozora se definiraju svi njegovi parametri, i objedinjuje sve naredne razine.

Druga razina je mreža (engl. *Grid*) u kojoj je smješten tok slike s Kinecta, kao i tok dubine i skeleta.

Zadnja razina, tipa platno (engl. *Canvas*) što označava spremnik vizualnih elemenata. Kako je ova razina nakon druge razine, ona je također dio prethodne razine, i njezini elementi su vizualno ispred elemenata prethodne razine. Unutar njega je smješten tok RGB video. Unutar ove razine nalaze se svi interaktivni gumbi (engl. *Button*), te natpisi (engl. *Label*).

Vizualno se lako mogu uočiti sve razine na slici koja je već prikazana (slika 24).

### 5.2.1 Gumbi za upravljanje

Interaktivni gumbi za upravljanje su kod otvorenog tipa i nalaze se unutar *Kinect4Fun* library paketa za C# programski jezik. Ovaj tip interaktivnog gumba je osmišljen upravo za Kinect senzor. Standardni gumb u Windows okruženju je polje koje ima svoje blobove[3], i kad korisnik dođe mišom u područje bloba on se može pokrenuti klikom miša. No, u slučaju Kincta, nema mogućnosti za klik miša. A aktivacija gumba samim dolaskom kursora-šake na područje bloba gumba, bilo bi dosta nespretno, jer korisnik na taj način može slučajno aktivirati gumb.

Zato je ugrađen interaktivni gumb. Njegova značajka je da kad korisnik dođe u područje globa šakom, oko šake se pojavi vijenac koja se puni 3 sekunde i kad se napuni tek tada se gumb aktivira. Da bi se vijenac napunila, korisnik mora biti sve vrijeme u području gumba, a ne točno u jednoj točki. Time se umanjilo korisnikovo naprezanje da održi svoju šaku u

[3] Blob je nakupina istovrsnih piksela (prema nekom kriteriju), termin se koristi u računalnoj obradi slike.

uskom području, nego mu je na taj način dano više prostora. Na slici (slika 25), je vidljiv interaktivni gumb prilikom punjenja kružnog vijenca.

**Slika 25 Prikaz punjenja interaktivnog gumba**

Upravo opisan tip gumba je lebdeći (engl. *Hover*) gumb. Prije tog rješenja, koristio se *magnet* gumb, no to rješenje je odbačeno prilikom evaluacije rada aplikacije. Naime, *magnet* gumb ima slična svojstva kao i *hover*, no bitna je razlika što on djeluje kao magnet za kursor šake. Tako, primjerice, kad se je korisnik našao u blizini gumba, gumb je kao magnet privlačio kursor i automatski ga dovukao u središte, nakon čega bi uslijedilo učitavanje akcije koje je trajalo 3 sekunde. Ni u ovom slučaju, korisnik nije morao držati ruku potpuno mirno, nego u radijusu oko gumba. Ovo rješenje u teoriji izgleda bolje nego *hover* gumb, no u praksi se pokazalo dosta nespretno za korisnika, a također i vrlo netočno, pošto je gumb hvatao kursor na nepredvidiv način, nekad vrlo blizu gumba koji je htio aktivirati, a nekad i puno dalje, pa je to bili zbunjujuće za korištenje.

U tablici (tablica 4) su prikazani svi elementi u razini *Canvas*, kako bi se dobio pregledan popis elemenata.

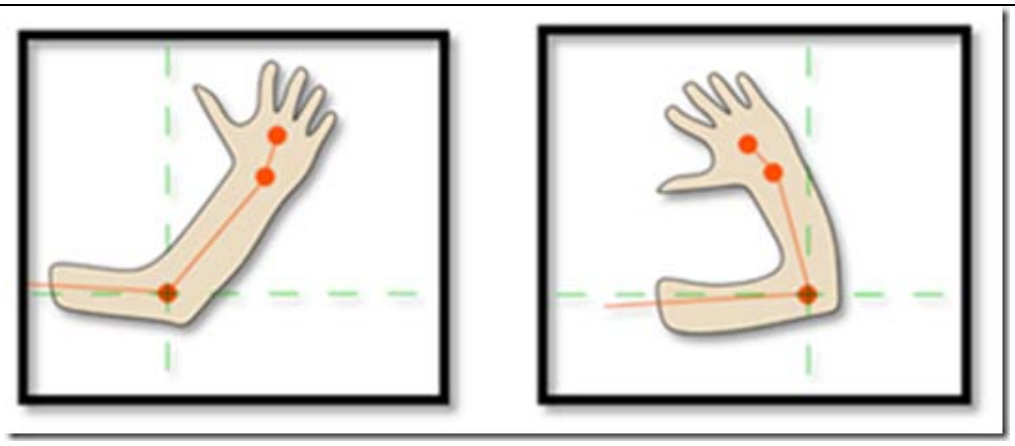
**Tablica 4 Popis vizualnih elemenata u aplikaciji**

| Broj | Ime elementa | Funkcija elementa | Tip elementa |
|---|---|---|---|
| 1 | HoverButton | Kursor šake | Kontrolni |
| 2 | label1 | Naslov aplikacije | Natpis |
| 3 | quitbutton | Izlaz iz aplikacije | Gumb |
| 4 | button_01 | Odabir lijevog spremnika | Gumb |
| 5 | button_02 | Odabir desnog spremnika | Gumb |
| 6 | button_1 | Odabir akcije za predmet kvadratni regulator u lijevi spremnik | Gumb |
| 7 | button_2 | Odabir akcije za predmet okrugli regulator a u lijevi spremnik | Gumb |
| 8 | button_3 | Odabir akcije za predmet kvadratni regulator druga strana u lijevi spremnik | Gumb |
| 9 | button_4 | Odabir akcije za predmet kvadratni regulator u desni spremnik | Gumb |
| 10 | button_5 | Odabir akcije za predmet okrugli regulator a u desni spremnik | Gumb |
| 11 | button_6 | Odabir akcije za predmet kvadratni regulator druga strana u desni spremnik | Gumb |
| 12 | button_N | Povratak s izbora predmeta na izbor spremnika | Gumb |
| 13 | txt_box | Upute korisniku | Natpis |

### 5.2.2 Implementirana gesta

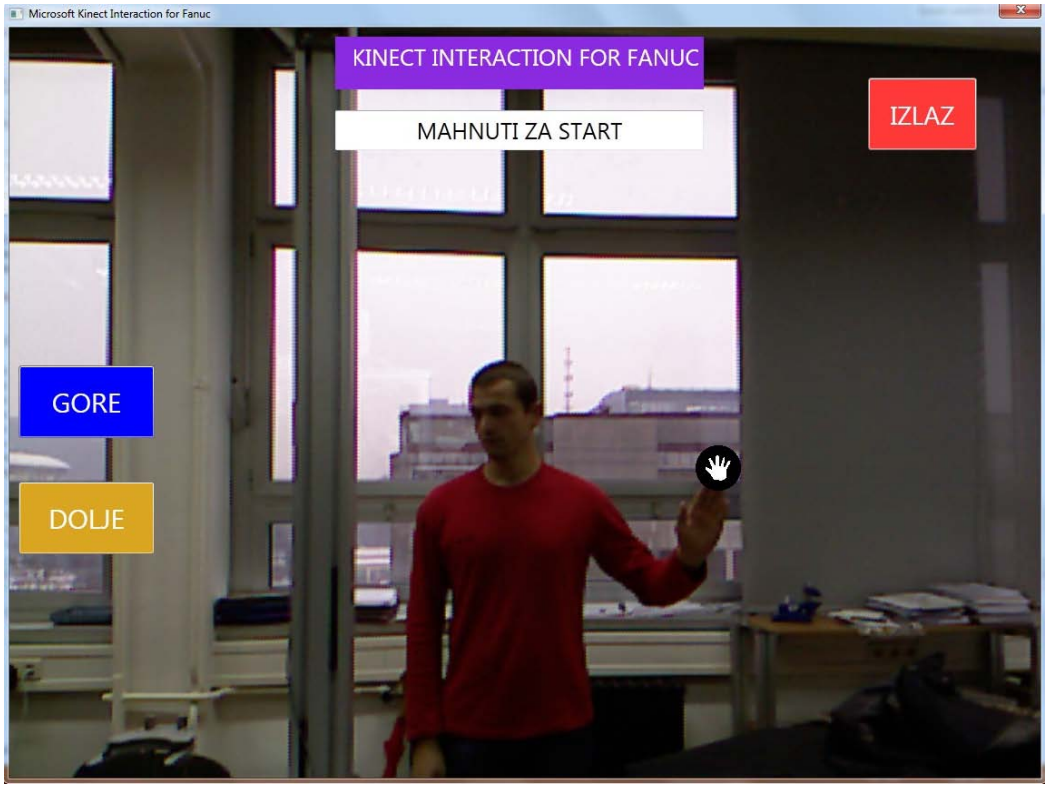
Gesta koja se koristi u radu je mahanje korisnika rukom. Da bi se prepoznala gesta, Kinect mora pratiti određene zglobove korisnika temeljem njihovih odnosa koji se mijenjaju u određenom vremenu. Za gestu mahanja rukom, Kinect prati 4 zgloba. Da bi se prepoznala gesta korisnik bi trebao nadlakticu imati na 90 stupnjeva +-10, a podlakticom mahati +. 45 stupnjeva. Potrebno je 2 puta mahnuti kako bi se detektiralo mahanje.

Bitno je za napomenuti da bi za uspješno prepoznavanje geste korisnik trebao nadlakticu imati otprilike podignutu za 90° od svoga trupa, kako je prikazano na slici (slika 26).



**Slika 26 Prikaz geste mahanja rukom [15]**

Gesta mahanja služi u da bi se pokrenuo robot, i time zapravo testiralo je li robot spreman za rad. Na slici (slika 27) se vidi aplikacija prije nego je korisnik pokrenuo vezu se robotom.




**Slika 27 Otvorena aplikacija**

## 5.3 Komunikacija s robotom

Kinect jednostrano komunicira s robotom preko aplikacije, što znači da samo šalje poruke preko TCP protokola. Dakako, kad se govori o slanju poruke robotu to je slanje poruke upravljačkoj jedinici robota koja obrađuje podatke i upravlja mjernim i mehaničkim sustavom robota.

Primanje poruka nije planirano za primjenu, ali bi dobro koristilo za stabilnost aplikacije. Primjerice, kada se izvodi radnja na robotu da se ne može slati naredba na njega, nego da po završetku radnje, robot signalizira aplikaciji da dopusti korisniku pokretanje neke druge radnje. Isto tako, moglo bi se signalizirati aplikaciji da je došlo do greške kad primjerice robot ne može vizijskim sustavom naći predmet rada i vrti se u beskonačnoj petlji.

Kako je komunikacija funkcionira i kako je postavljena, biti će više riječi u poglavlju 7.

# 6 Radnje s predmetima

U ovom poglavlju biti će riječi o praktičnoj strani rada koji se obavlja na radnom stolu od strane samog robota i vizijskog sustava. Valja napomenuti da je sustav djelomično uređen. Djelomično iz razloga što predmeti rada mogu biti bilo gdje na radnoj površini, i sustav će ih moći locirati. No sustav zapravo dosta uređen kada se pogleda činjenica da predmeti rada moraju biti na radnoj površini i još su ograničeni vidnim poljem kamere. Također, osvjetljenje mora biti zadovoljavajuće. Isto tako, spremnici moraju biti na fiksnim mjestima, pošto se raspolaže samo jednom kamerom, i robot ne može primijetiti da su spremnici pomaknuti.

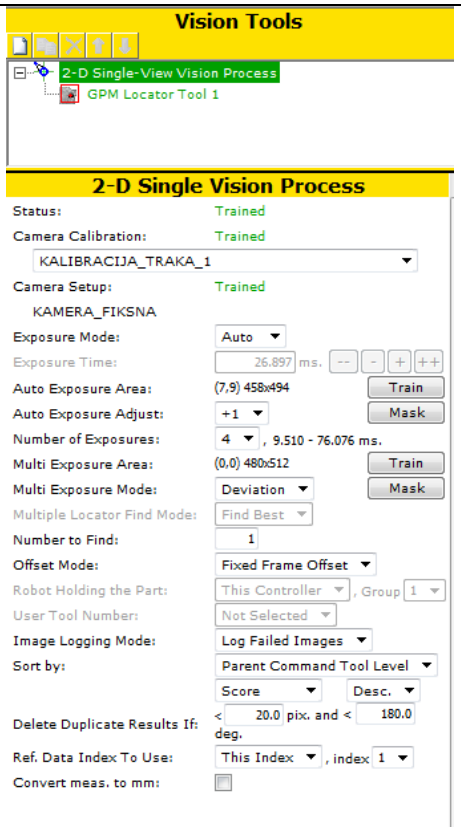
## 6.1 Vizijski sustav

Fanuc roboti imaju u sklopu svoje upravljačke jedinice podršku za vizijske procese obrade slike. Točnije, obrada slike se izvodi u upravljačkoj jedinici, ali se sve prikazuje na računalu, kako bi korisnik imao mogućnost upravljanja njime.

Vizijski proces je sustav koji služi robotu da može korištenjem kamere prepoznati od prije naučen predmet rada, naći njegovo težište, konturu i značajke, te omogućiti robotu da može prići predmetu i izvršiti određenu radnju.

Vizijski proces je puno korišten u radu, i bitan je dio pošto je krucijalni aspekt rada „*pick and place*“ kojemu je potrebna pozicija i orijentacija predmeta, a koju robot ne može znati ako nema podatke od nekog vanjskog uređaja, u ovom slučaju, kamere. Nadalje će biti pojašnjen sam vizijski proces, i kako je on implementiran u radu.

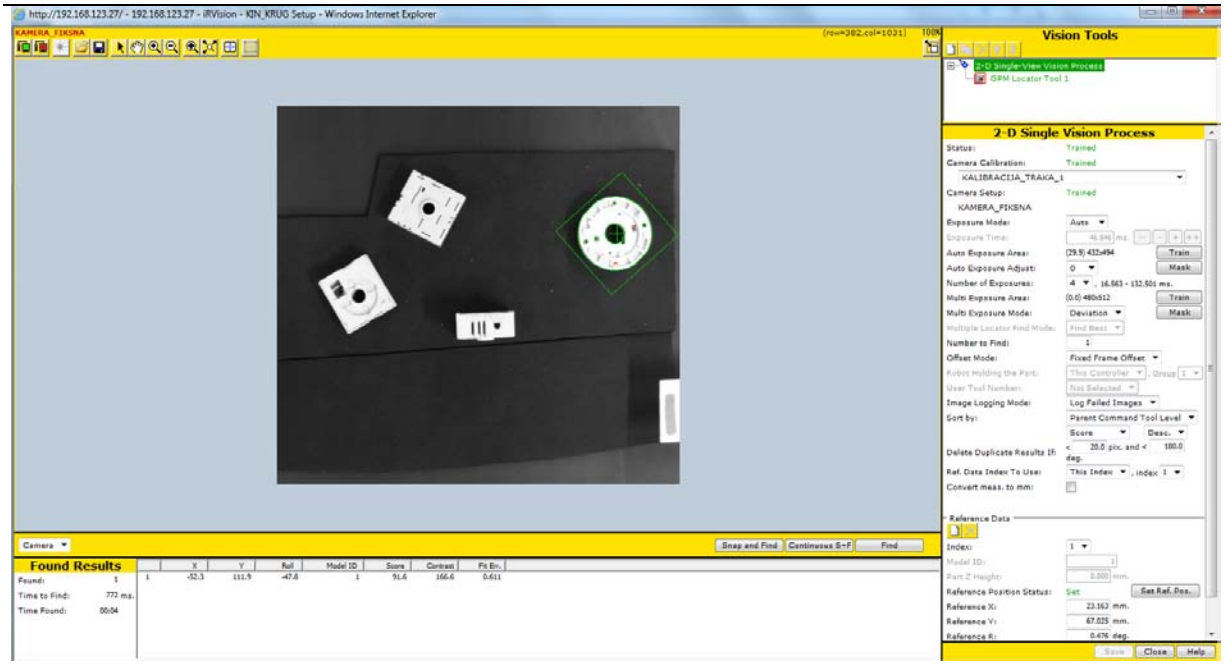
Vizijski proces funkcionira tako da se najprije načini kalibracijska ravnina, pomoću koje robot zna na kojoj je visini (u kojoj ravnini) predmet. Na slici (slika 28) se vidi dijaloški okvir koji tome služi. Kalibracijska ravnina je napravljena otprije, i korištena je kao gotova za ovaj rad.




**Slika 28 Odabir kalibracijske ravnine**

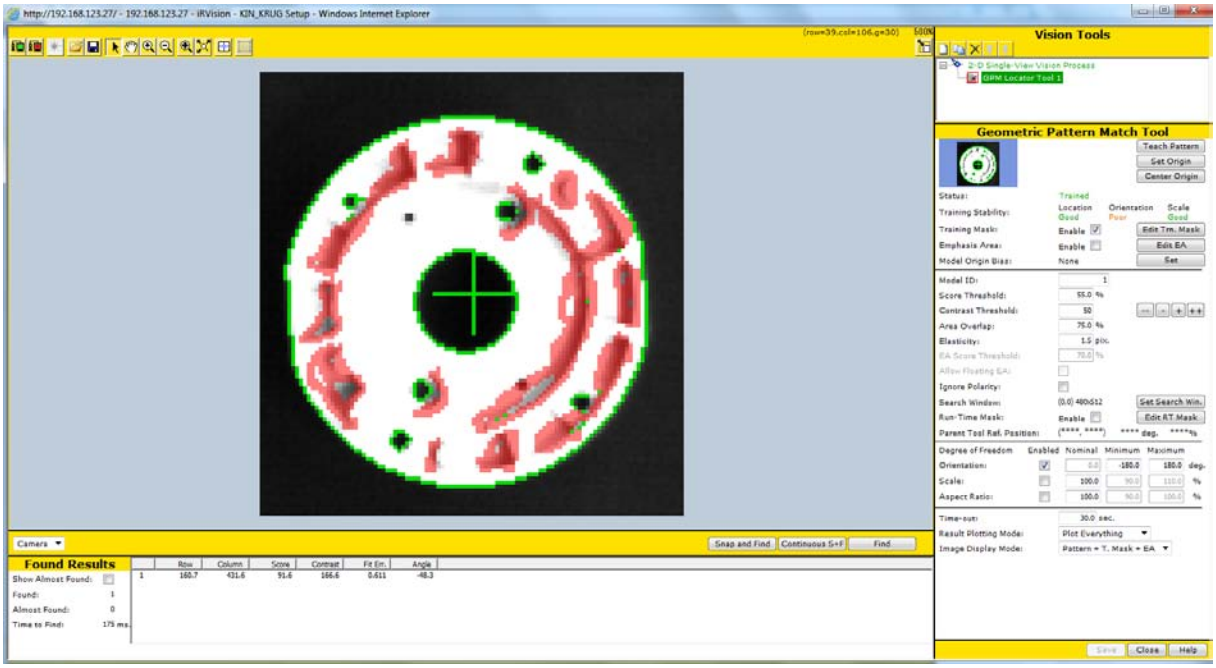
Slijedeći korak je postaviti predmet na kalibracijsku ravninu i slikati predmet. Nakon što program ima sliku koju je slikala kamera, može se dalje pristupiti obradi slike, na način da se podešavaju parametri, a najbitniji su oni koji utječu na svjetlost. Na slici 28 se ti parametri mogu primijetiti u središnjem dijelu, to su Način ekspozicije, koji je postavljen na automatski način rada To znači da će se prilikom svakog slikanja kamere ekspozicija namještati za +1 u konkretnom slučaju. Nadalje, treba robotu ukazati koji je njegov predmet rada na slici, i to tako što se jednostavno se robota nauči uzorak tog predmeta.

Na slici (slika 29) je prikazan isječak koji pokazuje dijaloški prozor koji se otvara nakon slikanja kamere i pronalaženja predmeta.




**Slika 29 Primjer pronalaska naučenog predmeta**

Da bi sustav došao do te faza kada može prepoznati naučeni predmet, mora ga se naučiti kako da to napravi. Sustav funkcionira tako da vizijski sustav detektira konture predmeta i blobove, a na korisniku je da odabere što je bitno za točno pozicioniranje i orijentiranje hvataljke robota, neovisno gdje se nalazi premet rada u budućem radu sustava. To je prikazano na slici (slika 30).




**Slika 30 Primjer selekcije pronađenih kontura na predmetu**

Crvena područja su ručno označene značajke koje su označene kako ih sustav ubuduće ne bi uzimao u obzir kod detekcije predmeta.

Još preostaje snimiti referentnu točku robota koja će služiti za pozicioniranje hvataljke u odnosu na predmet rada tako da ga robot u može dohvatiti u svakoj orijentaciji. To se inače radi tako da se pronađe predmet u vizijskom procesu i nakon toga se robot dovede u položaj koji je najpogodniji da bude referentni, tj. onaj položaj od kojeg vizijski sustav prilikom svakog budućeg pronalaženja predmeta računa odmak, i taj parametar služi za točno postavljanje kalibriranog alata u odnosu na predmet rada, kako bi ga mogao uspješno zahvatiti.

Za komunikaciju robota i programske podrške za viziju na računalu, koristi se standardni IP protokol.

## 6.2 Detektiranje predmeta rada i implementacija u sustav

Detekcija predmeta se vrši korištenjem vizijskog sustava, točnije, procesa, za svaki pojedini predmet, kako je već ranije spomenuto. No, potrebno je bilo implementirati u samu aplikaciju. To je riješeno na način da korisnik interakcijom sa gumbom pokrene program koji je već spremljen u robota, a unutar njega se nalazi vizijski proces.

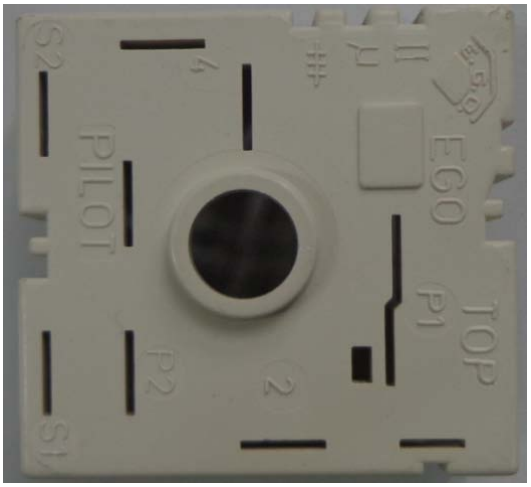
Jedina mana ovakvog pristupa je problem koji je se javlja kada iz bilo kojeg razloga vizijski proces ne detektira predmet rada. Taj segment nije obuhvaćen u radu zbog nedostatka vremena, i može dobro doći kao buduća nadogradnja. Jedan od načina na koji bi se moglo doskočiti tom problemu je da robot pošalje poruku u aplikaciju u računali da je došlo do greške, ili da ispiše poruku na *i* Pendantu u polju „USER“ da je došlo do problema u radu programa.

Za reprezentativnost rada koristit će se uzmi i odloži radnja. Kako se koristi vizijski sustav, predmete je moguće postaviti unutar vidnog područja kamere u bilo koju poziciju, a robot će ih moći prepoznati i uhvatiti. Sve se radi pod pretpostavkom da je osvjetljenje u radnom prostoru zadovoljavajuće, pa vizijski sustav može locirati predmet. Ukoliko predmeta s kojim se želi baviti radnja nema, program će pokušati pronaći neki drugi predmet od ponuđena 3 kako program ne bi ušao u petlju iz koje ne može izaći. Ako nema nijednog predmeta koji su naučeni, program će stati i morat će se ponovo pokrenuti.

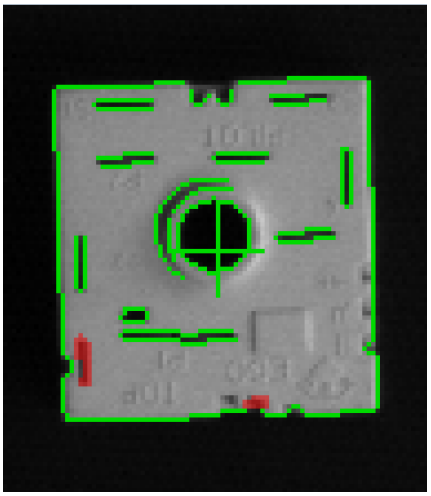
U radu se radi s 2 različita predmeta rada, no zapravo su tri, pošto je jedan različit s prednje i stražnje strane, pa kako bi se omogućilo da bude potpuno nebitno kako se on postavlja na radno područje, u aplikaciji postoje 3 predmeta.

Riječ je o kućištima i spojnim dijelovima za elektro industriju.

Na slici (slika 31) se vidi premet kućište termo regulatora, i to svojoj prvoj varijanti. Na slici (slika 32) se vide značajke i rubovi predmeta koje je prepoznao vizijski proces, gdje se vidi crvenom bojom što je zanemareno prilikom učenja vizijskog sustava da prepozna predmet.



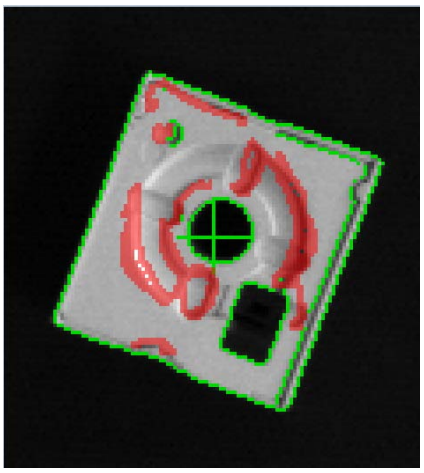

**Slika 31 Predmet rada - kućište**



**Slika 32 Značajke za prepoznavanje predmeta kućište**

Na slici (slika 33) se vidi predmet kućište termo regulatora, ali sa strane poklopca, čija se topologija uvelike razlikuje, kada se slika s ove strane. S ove strane, kućište ima kružni otvor, ali ima i otvor sa strane. Zato je vizijski proces napravljen posebno za ovu orijentaciju predmeta pa tako egzistiraju 2 predmeta u vidu operacije s njima. Na slici (slika 34) se može vidjeti dio vizijskog procesa prilikom učenja prepoznavanja za ovaj predmet, a može se zamijetiti da su sada različite značajke u vizijskom procesu zanemarene.

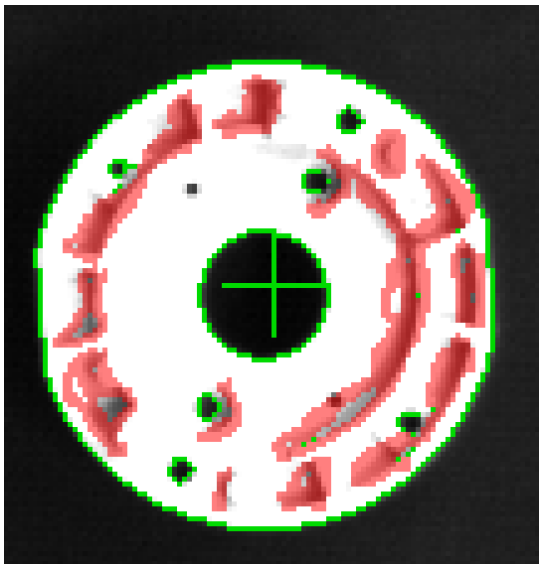
**Slika 33 Predmet rada - poklopac**



**Slika 34 Značajke za prepoznavanje predmeta kućište 2**

Zadnji predmet je okruglo kućište termo regulatora, koji ima slične značajke na obje strane pa se ukazala mogućnost primjene samo jednog vizijskog procesa. Na slici (slika 35) se može vidjeti slika predmeta, a na slici (slika 36) vizijski proces za predmet. Vidi se da su neke od značajki koje pronalazi vizijski proces zanemarene, kako bi se omogućilo prepoznavanje predmeta i s druge strane, a druga strana prikazana je slikom (slika 37).

**Slika 35 Predmet rada - okruglo kućište**



**Slika 36 Značajke za prepoznavanje predmeta – okruglo kućište**

Za predmet rada na slici (slika 37) vizijski proces je potpuno isti kao i za predmet rada na slici (slika 35), pošto je mnogo značajki koje su prepoznate, eliminirano, kako bi se mogao jednoznačno prepoznati predmet i s druge strane.

**Slika 37 Okrugli regulator s druge strane**

Kako u postavkama vizijskog procesa postoji mogućnost za određivanje postotka podudarnosti naučenog predmeta i predmeta koji se pojavi prilikom iduće radnje. Za predmet okrugli regulator je taj postotak postavljen na 60 %, kako bi proces mogao prepoznati i predmet s druge strane, i nije potreban dodatni vizijski proces.

## 6.3 Uzmi i odloži

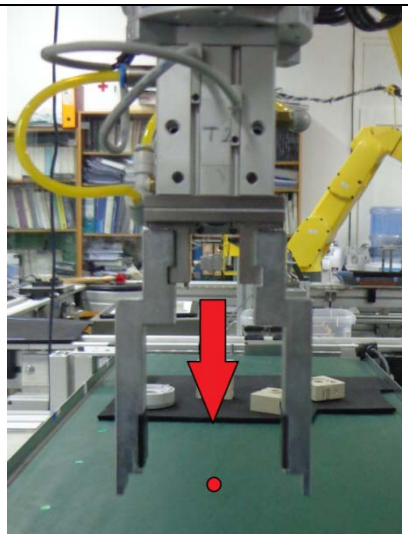
Uzmi i odloži radnja upravo je karakteristična za robote heksapodne strukture, zbog svoje jednostavnosti. Radnja se može razložiti na 3 djela.

Prvi dio je pozicioniranje robota u položaj gdje kamera može slikati cijelim vidnim poljem područje gdje se nalaze predmeti, te slikanje i detekcija predmeta.

Drugi dio je slanje robotu pozicije i orijentacije predmeta, kako bi on mogao krenuti i doći u točku hvatanje predmeta. Kada robot zahvati predmet, on bi ga trebao vertikalno podići.

Konačno treći dio je odlaganje predmeta u zahvatu na određenu poziciju.

Za pravilno izvođenje operacija s alatom, u konkretnom slučaju, hvataljkom, potrebno ju je kalibrirati. Kalibracija alata služi da se može koristiti koordinatni sustav alata kao osnovni za gibanja robota. Obično se kod kalibracije hvataljke uzima točka kalibracije tako da bude smještena na sredini, između 2 kraka hvataljke, pri samom vrhu alata. To se može vidjeti na slici (slika 38).

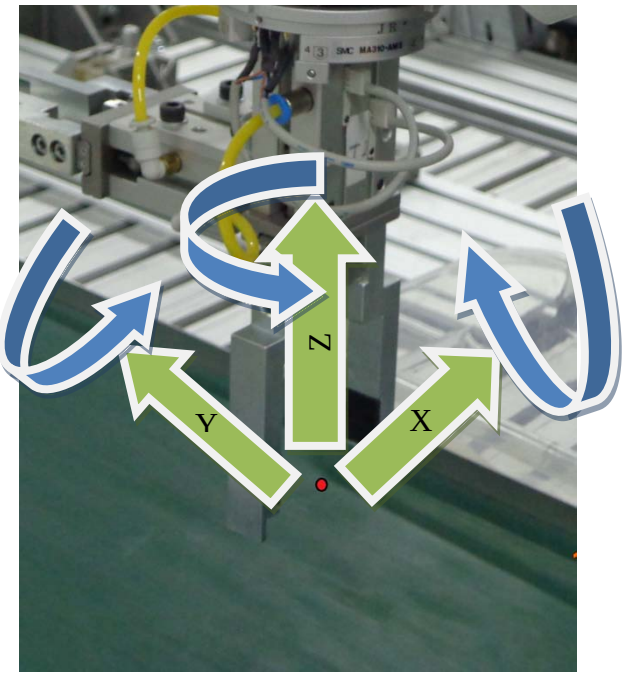


**Slika 38 Točka za kalibraciju alata**

Koordinatni sustav koji se koristio prilikom pisanja programa je koordinatni sustav alata hvataljke robota. Za potrebe rada, korištena je kalibracija koja se može vidjeti na slici (slika 39).

Najviše je korišten koordinatni sustav alata, upravo zbog rotacije hvataljke, no robot ima mogućnost korištenja još 3 koordinatna sustava. Jedan je koordinati sustav svijeta, drugi je koordinatni sustav samog robota, a treći je proizvoljni koordinatni sustav korisnika.

Na slici (slika 40) je prikazan koordinatni sustav alata koji je korišten prilikom pisanja svih programa u *i*Pendantu Sustav je desnokretan, pa strelice prikazuju pozitivnu translaciju/rotaciju.




**Slika 39 Koordinatni sustav alata**

Programi za svaki pojedini predmet i svaki pojedini spremnik su pisani izravno u *i*Pendantu. Tako da je broj programa 6, a on je proširen još trima dodatnim vizijskim procesima koji mogu prepoznati predmet kvadratno kućište ako je okrenuto na bočnu stranu. Pa se u konačnici koristi 6 programa sa 6 vizijskih procesa koji se pozivaju prilikom rada sustava, a svi moraju biti upisani u *i*Pendant, kako bi se mogli pravovremeno pozivati.
Njihova imena i funkcije prikazani su tablicom (tablica 5).

**Tablica 5 Popis programa koji se pozivaju**

| Ime programa | Glavni vizijski proces koji se koristi | Predmet rada | Funkcija |
|---|---|---|---|
| **DOD_1** | KIN_KRUG | Okrugli regulator | *Pick and place* u desni spremnik |
| **DOD_2** | KIN_POKLOPAC | Kvadratni regulator | *Pick and place* u desni spremnik |
| **DOD_3** | KIN_KUCISTE | Kvadratni regulator | *Pick and place* u desni spremnik |
| **PICK_1** | KIN_POKLOPAC | Kvadratni regulator | *Pick and place* u lijevi spremnik |
| **PICK_2** | KIN_KRUG | Okrugli regulator | *Pick and place* u lijevi spremnik |
| **PICK_3** | KIN_KUCISTE | Kvadratni regulator | *Pick and place* u lijevi spremnik |
| | KIN_BOK1 | Kvadratni regulator | *Pick and place* u lijevi ili desni spremnik |
| | KIN_BOK2 | Kvadratni regulator | *Pick and place* u lijevi ili desni spremnik |
| | KIN_BOK3 | Kvadratni regulator | *Pick and place* u lijevi ili desni spremnik |

Suština programa koju su napisani u *i*Pendantu je to da se radnje izvršavaju kad se prepozna predmet. Za predmet kvadratnog regulatora, u slučaju da je okrenut bočno na radnoj površini, napravljena su 3 vizijska procesa, pošto su 3 od 4 bočne stranice vrlo različite, a dvije su vrlo slične, pa se može riješiti dvije strane s jednim vizijskim procesom. Pa tako primjerice, kad

korisnik pozove primjerice p*ick and place* u desni spremnik za predmet kvadratni regulator sa strane poklopca, a takvog predmeta nema na radnoj površini, algoritam naređuje vizijskom procesu da proba naći kvadratni regulator okrenut na bok, pa se pozivaju unutar programa DOD_2 pozivaju vizijski procesi KIN_BOK1, KIN_BOK2, KIN_BOK3, pa ako niti jedan od njih ne pronađe regulator okrenut na bok, tada program tek staje. U slučaju da ga pronađe na boku, uzima predmet i stavlja ga u desni spremnik.

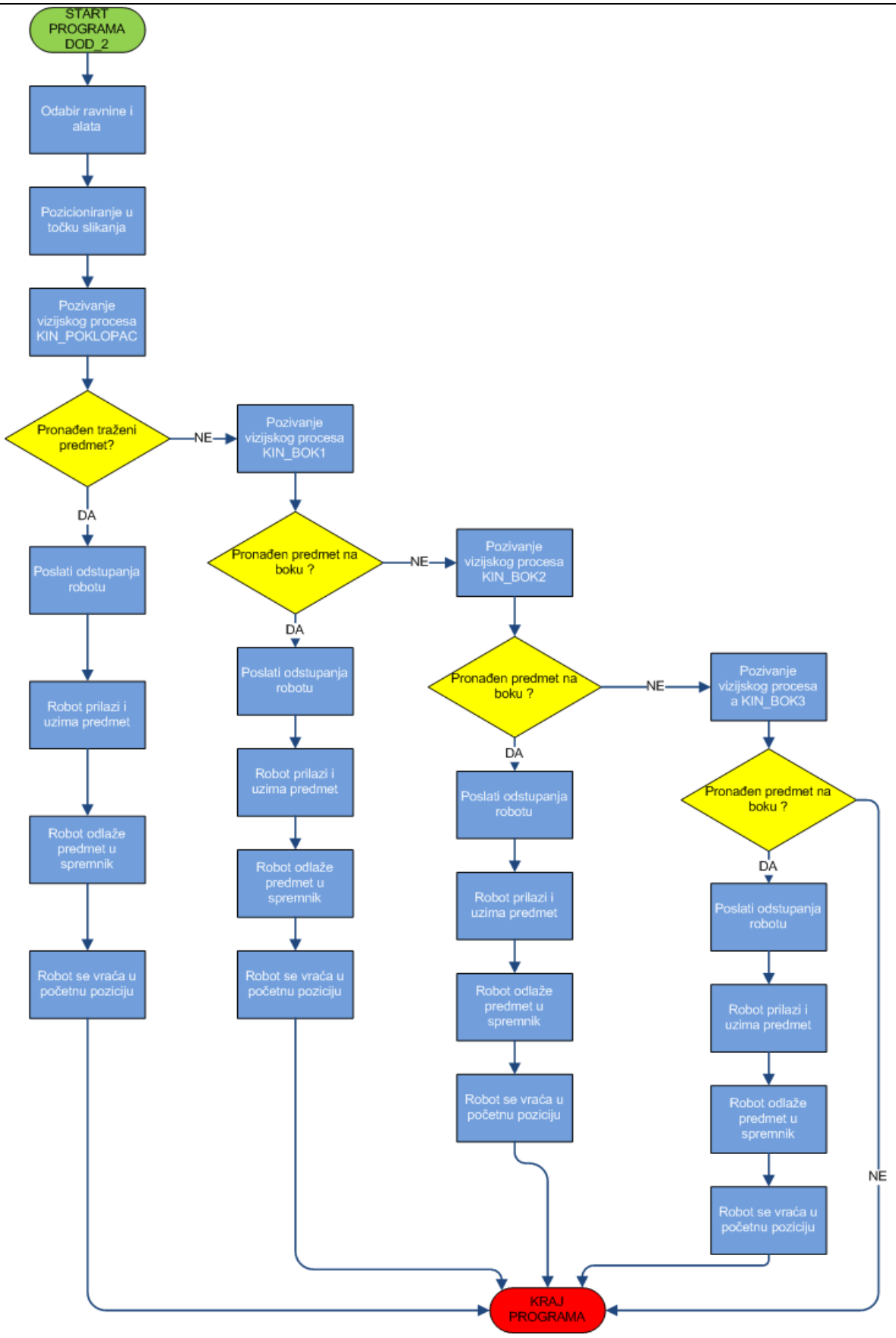
Da bi se lakše shvatilo kako programi funkcioniraju, prikazat će se redoslijed operacija kako slijede u samom programu u *i*Pendantu za program DOD_2, da se skrati prikaz, prikazano je u dvije kolone.

```
UFRAME_NUM =1
UTOOL_NUM=3
CALL GR_OP
!****
VIZIJSKI PROCES
!*****
P[1:TOCKA SLIKANJA] 100mm /sec,
FINE VISION RUN_FIND
„KIN_POKLOPAC“
VISION GET_OFFSET
„KINECT_POKLOPAC“ VR [1] JMP
LBL[20]
!****
PRIHVAT PREDMETA 1
!******
P [2:TOCKA UZIMANJA]  100mm /sec,
FINE
FINE VOFFSET, VR[1]
TOOL_OFFSET,PR[1]
P[2 :TOCKA UZIMANJA] 100mm/sec,
FINE
FINE VOFFSET,VR[1]
CALL GR_CL
P [2:TOCKA UZIMANJA]  100mm /sec,
FINE
FINE VOFFSET, VR[1]
TOOL_OFFSET,PR[1]
!****
ODLAGANJE 1
!****
P[3: TOCKA ODLAGANJA]  100mm/sec
FINE TOOL_OFFSET, PR[1]
P[3: TOCKA ODLAGANJA]  100mm/sec
FINE
CALL GR_OP
P[3: TOCKA ODLAGANJA]  100mm/sec
FINE TOOL_OFFSET, PR[1]
P[1:TOCKA SLIKANJA] 100mm /sec,
FINE
!****KRAJ****
END
LBL [20]
VISION RUN_FIND „KIN_BOK“
VISION GET_OFFSET „KIN_BOK“ VR [20]
JMP LBL[30]
!****
PRIHVAT PREDMETA2
!******
P [2:TOCKA UZIMANJA]  100mm /sec,
!****
ODLAGANJE 3
!****
FINE
FINE VOFFSET, VR[1]
TOOL_OFFSET,PR[1]
P[2 :TOCKA UZIMANJA] 100mm/sec,
FINE
FINE VOFFSET,VR[1]
CALL GR_CL
P [2:TOCKA UZIMANJA]  100mm /sec,
FINE
FINE VOFFSET, VR[1]
TOOL_OFFSET,PR[1]
!****
ODLAGANJE 2
!****
```

```
P[3: TOCKA ODLAGANJA]  100mm/sec
FINE TOOL_OFFSET, PR[1]
P[3: TOCKA ODLAGANJA]  100mm/sec
FINE
CALL GR_OP
P[3: TOCKA ODLAGANJA]  100mm/sec
FINE TOOL_OFFSET, PR[1]
P[1:TOCKA SLIKANJA] 100mm /sec,
FINE
!****KRAJ****
END
LBL [30]
VISION RUN_FIND „KIN_BOK2“
VISION GET_OFFSET „KIN_BOK2“ VR [1]
JMP LBL[40]
!****
PRIHVAT PREDMETA 3
!******
P [2:TOCKA UZIMANJA]  100mm /sec,
FINE
FINE VOFFSET, VR[1]
TOOL_OFFSET,PR[1]
P[2 :TOCKA UZIMANJA] 100mm/sec,
FINE
FINE VOFFSET,VR[1]
CALL GR_CL
P [2:TOCKA UZIMANJA]  100mm /sec,
FINE
FINE VOFFSET, VR[1]
TOOL_OFFSET,PR[1]

P[3: TOCKA ODLAGANJA]  100mm/sec
FINE TOOL_OFFSET, PR[1]
P[3: TOCKA ODLAGANJA]  100mm/sec
FINE
CALL GR_OP
P[3: TOCKA ODLAGANJA]  100mm/sec
FINE TOOL_OFFSET, PR[1]
P[1:TOCKA SLIKANJA] 100mm /sec,
FINE
!****KRAJ****
END
LBL [40]
VISION RUN_FIND „KIN_BOK3“
VISION GET_OFFSET „KIN_BOK3“ VR [1]
JMP LBL[99]
!****
PRIHVAT PREDMETA 4
!******
P [2:TOCKA UZIMANJA]  100mm /sec,
FINE
FINE VOFFSET, VR[1]
TOOL_OFFSET,PR[1]
P[2 :TOCKA UZIMANJA] 100mm/sec,
FINE
FINE VOFFSET,VR[1]
CALL GR_CL
P [2:TOCKA UZIMANJA]  100mm /sec,
FINE
FINE VOFFSET, VR[1]
TOOL_OFFSET,PR[1]
!****
ODLAGANJE 4
!****
P[3: TOCKA ODLAGANJA]  100mm/sec
FINE TOOL_OFFSET, PR[1]
P[3: TOCKA ODLAGANJA]  100mm/sec
FINE
CALL GR_OP
P[3: TOCKA ODLAGANJA]  100mm/sec
FINE TOOL_OFFSET, PR[1]
LBL [99]
P[1:TOCKA SLIKANJA] 100mm /sec,
FINE
!****KRAJ****
END
```

U gore navedenom prikazu koda, može se zamijetiti korištenje zastavica (engl *Label*) koji služe za preskakanje određenog dijela koda. Koriste se vizijski (VR [1]) i pozicijski registri (PR[1]) kako bi se skratila dužina samog koda, i smanjio broj točaka koje robot mora upamtiti. Valja napomenuti da je rad s registrima jednostavan, no treba pripaziti i provjeriti što se u njima nalazi prije svakog korištenja robota, pošto ih netko drugi može izmijeniti za svoje potrebe.




**Slika 40 Blok dijagram za program s 4 vizijska procesa**

U prethodnom dijagramu (slika 40) može se pratiti primjer programa DOD_2 koji je opisan u paragrafu iznad, dakle *pick and place* kvadratnog regulatora.

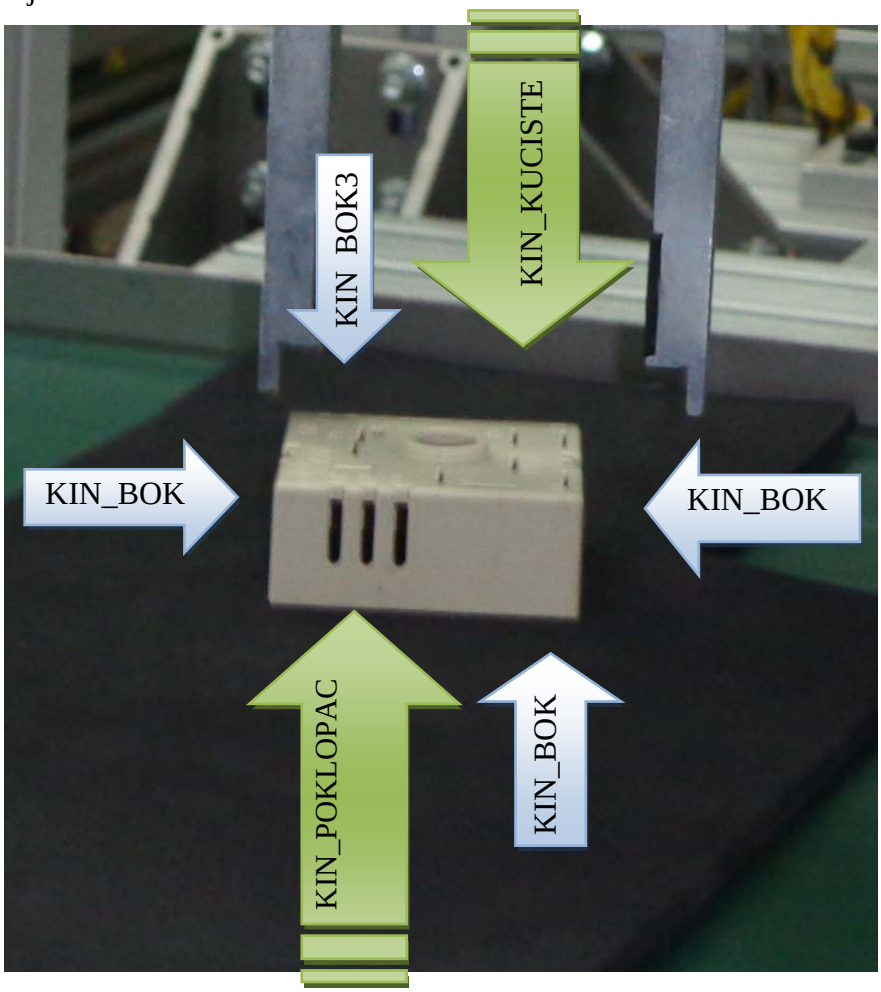
U programu DOD_3, također egzistiraju 4 vizijska procesa od kojih je glavni KIN_KUCISTE.

Kod programa DOD_1, za rad s okruglim regulatorom, postoji samo jedna vizijski proces, koji je dostatan za predmet kada je okrenut s gornje ili donje strane.

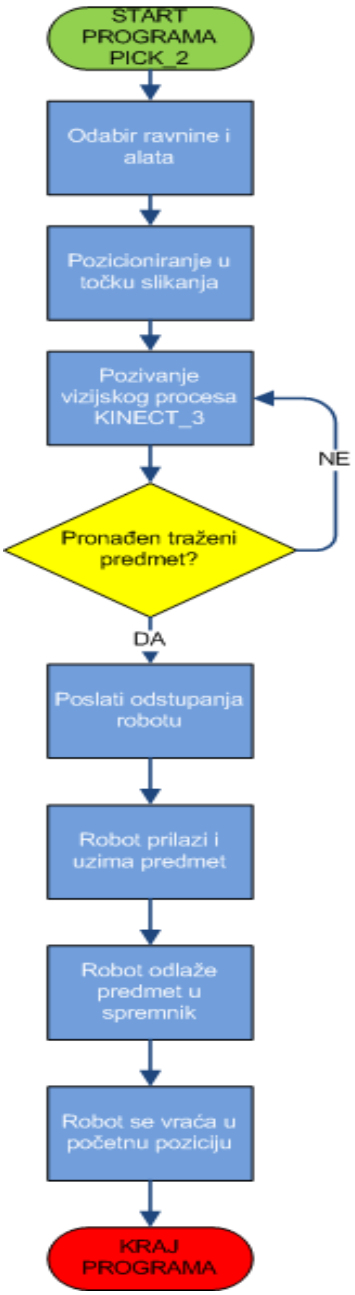
Analogno kao i za spremnik s dese strane, i spremnik s lijeve strane ima svoja 2 programa za kvadratne regulatore PICK_1 za regulator sa gornje strane , te PICK_3 za regulator sa strane poklopca, te jedan program za okrugli regulator PICK_2.

Programi PICK_1 i PICK_3 su analogni, kao i za desni spremnik, dakle, imaju po 4 vizijska procesa.

Da bi se zornije prikazalo s koje strane vizijski proces snima pojedini predmet, napravljana je slika (slika 41) koja prikazuje strelicama smjer kako su predmeti položeni na radnoj površini za vrijeme rada ovisno o vizijskom procesu. Razumljivo je da su bočne stranice kraće, i one su prikazane bijelim strelicama.




**Slika 41 Prikaz svih vizijskih procesa za pravokutni termo regulator**




**Slika 42 Blok dijagram za program s jednim vizijskim procesom**

Za predmet rada – okrugli regulator, su predviđene samo radnje kada je predmet okrenut na jednu ili drugu okruglu stranu, pošto ne može zbog kružnog oblika biti na boku, a i ograničenje, visine hvataljke limitira visinu predmeta koji se zahvaća.

Na slici (slika 42) je prikazan blok dijagram za program PICK_2 koji služi za radnju *pick and place* okruglog regulatora. Koristi se samo jedan vizijski proces, koji služi za radnje s jedne ili druge strane regulatora (slika 35 i slika 37) u pod poglavlju 6.2.

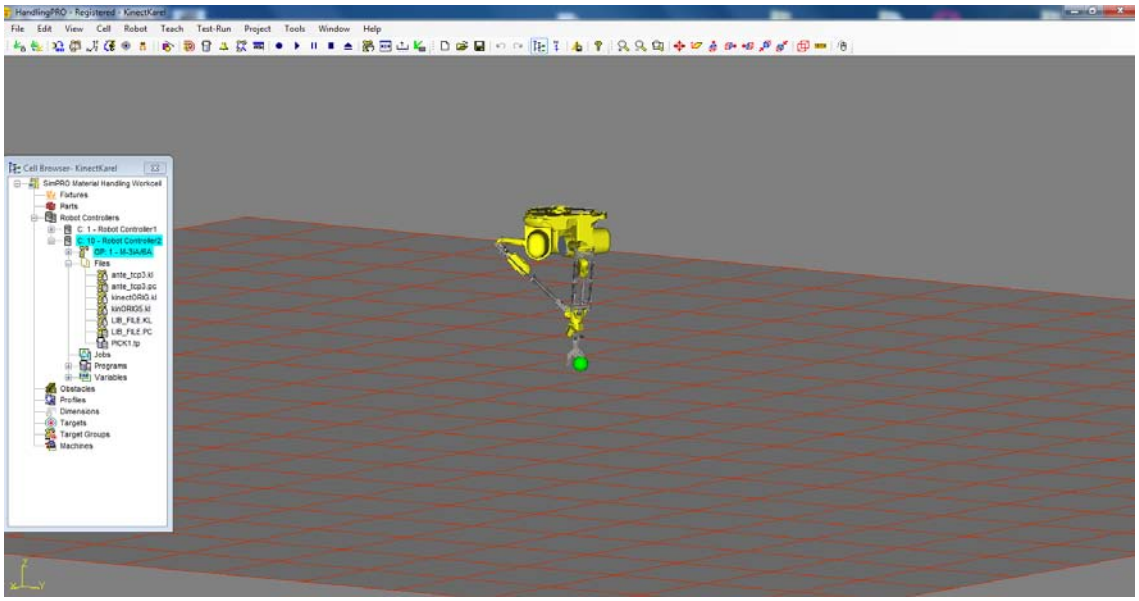
# 7 Veza aplikacije i robota

Komunikacija se vrši jednosmjerno, i to tako da aplikacija šalje naredbe robotu koje akcije da poduzima. Jedini problem koji bi se mogao dogoditi na robotu jest vizijski sustav, a obavijesti glede njega su implementirane unutar programa na *i*Pendantu, i korisnik vidi koji je program pokrenuo i tako može kontrolirati rad.

## 7.1 Karel

Karel je programski jezik koji koriste Fanuc roboti za programiranje gibanja i ostalih radnji unutar softverske podrške u vidu programa Roboguide. Pod programiranjem u kontekstu robotike se podrazumijeva da je to veza između rješenja problema i upravljačkog sustava robota. To je niži programski jezik koji je orijentiran gibanju , za razliku od objektnih programskih jezika. Samim time pojedine algoritme teško je implementirati, no za vrlo složene algoritme današnji industrijski roboti nisu namijenjeni.

Da bi se počelo programirati, potrebno je načiniti unutar Roboguide-a robotsku ćeliju sa potrebnim parametrima, kao što su CAD model robota i njegova postavke, alat robota, kinematiku te broj upravljanih osi. Kada je to uspješno napravljeno, prikazuje se robotska ćelija s robotom ili više njih, kao što je vidljivo na slici (slika 43).



**Slika 43 Isječak iz programa Roboguide**

Nakon toga se može krenuti s programiranjem u programskom jeziku karelu, a ukoliko će se raditi sa stvarnim robotom, a ne modelom unutar Roboguide programa, potrebno je spojiti

robota s računalom, kako bi Roboguide mogao s njim razmjenjivati podatke. Kada se kaže robot u kontekstu veze se računalom ili bilo kojim vanjskim uređajem, uvijek se misli na upravljačku jedinicu robota, koja obrađuje sve podatke koji se šalju na robota, ili koje on šalje spojenom uređaju.

Samo programiranje se vrši u nekoliko koraka. Prvi korak se sastoji od pisanja koda koji se upisuje u .kl datoteku. Funkcije i naredbe koji se mogu implementirati su raznovrsni i orijentirani na općenite potrebe robota. Za shvaćanje osnovnih metoda za programiranje korištenih u radu, korišten je priručnik Fanuc R-30iA Controller KAREL Reference Manual.

Kada je završeno pisanje koda u .kl datoteku, ona se mora spremiti, a nakon toga prevesti u strojni jezik upravljačkog sustava robota. Ukoliko nije dosljedno slijeđena sintaksa ili se potkrala graška kod kodiranja u .kl datoteci, neće biti moguće prevođenje u strojni jezik i potrebna je revizija. Nažalost, Roboguide ne nudi opsežnu pomoć pa treba detaljno pregledati korisničke upute.

Kada je nastupila faza uspješnog prevođenja u strojni jezik upravljačkog sustava robota, Roboguide stvara .pc datoteku. Ona se može poslati na robota koji mora biti spojen TCP vezom s računalom i evidentiran u popisu robota ili se učita izravno u Roboguide sučelje za simulaciju gibanja. Na žalost, Roboguide nema podršku za simuliranje TCP veze, pa je u slučaju rada program uvijek poslan na robota, točnije u njegovu upravljačku jedinicu.

## 7.2 Primjena u radu

Za konkretan rad, Karel je bio potreban za ostvarivanje komunikacije između upravljačke jedinice robota i aplikacije na računalu. Taj dio programiranje se ne može izvesti na *i* Pendantu jer on samo služi za programiranje gibanja i kontrole robota od strane korisnika, a ujedno i za pokretanje programa koji su prethodno napisani u Karelu unutar Roboguide programa.

Koriste se 2 programa. Oba moraju biti učitani u *i*Pendant, kako bi mogli raditi.

Jedan je library datoteke imena LIB_FILE, koji sadrži osnovne rutine za komunikaciju preko socketa koji se koriste za komunikaciju robota, a to su:

- OPEN_FILE

  Open file rutina služi za otvaranje datoteke u upravljačkoj jedinici robota. Ona se poziva u glavnom programu na početku, odmah nakon deklariranja varijabli, kako bi se odmah mogle primati ili slati poruke.

- CLOSE_FILE

  Close file rutina služi za zatvaranje otvorenih datoteka u upravljačkoj jedinici robota, i većinom se koristi na kraju programa, ili ako se želi s namjerom zatvoriti otvoreni file tijekom rada programa.
- HANDSHAKING

  Handshaing rutina se koristi kad se šalju poruke između 2 ili više robota spojenih u mrežu, i ima parametre kao što su IP adresa pojedinog robota, i ostalo što je potrebno. Ova rutina nije korištena u radu.

Cjeloviti ispis datoteke LIB_FILE nalazi se u prilogu rada.

Drugi program (KIN_ORIG5) je čisti program koji se pokreće prije pokretanja same aplikacije na računalu. On služi tome da prima poruke od strane aplikacije i šalje ih upravljačkoj jedinici, tj. robotu, što se manifestira aktiviranjem programa na robotu i obavljanjem zadane radnje. U sebi sadrži rutine iz library datoteke koje se pozivaju za rad programa. Glavni dio programa je čitanje poruka iz aplikacije i njihovo interpretiranje. Nakon toga, pozivanje programa za rad s predmetima, unutar kojih su sve potrebne naredbe da bi se ostvarila tražena akcija.
Cjeloviti ispis datoteke KIN_ORIG5 nalazi se u prilogu rada.

IP adresa robota u mreži robota koja je instalirana u laboratoriju je 192.168.1.27, dok je njegov port za komunikaciju 5555, što se može vidjeti u prilogu 1.
Prilikom otvaranja adrese u internet pregledniku, otvara se server upravljačke jedinice robota gdje se mogu administrirati sve potrebitosti vezane uz rad.

# 8 Testiranje rada sustava

Nakon izrade aplikacije i programa na robotu, te uspostave veze aplikacije i robota, cijeli rad treba biti ispitan.

Prilikom testiranja i ispitivanja, javilo se mnoštvo grešaka i problema koji su uklonjeni na različite načine.

Prvi problem se javio prilikom stvaranja aplikacije, točnije, kada je implementiran u sustav kursor u obliku šake koji je pratio korisnikovu najbližu šaku. Javljala se je velika nestabilnost koja se očitovala u velikom odmaku prave pozicije šake korisnika i kursora. Pošto se je ta pojava javljala tako što je odmak uvijek bio na lijevu stranu šake, pokušalo se promjenom rezolucije glavnog prozora i rezolucije sloja koji prikazuje sliku s Kinecta postići kompromis, no tada se javljala pojava da se šaka korisnika i kursor poklapaju, no označavanje željenog gumba je postao problem i javio se još veći odmak.

Konačno je cijeli kod izmijenjen i postavljen novi tip gumba – *hover* gumbi,umjesto *magnetic* gumba koji su prije bili implementirani i imali su spomenute loše implikacije u sustavu.

Druga prepreka bila je veza upravljačke jedinice robota i aplikacije. Naime, kako se postupno gradila aplikacija, preskočeno je testiranje uspostavljanja veze. No, kada je došlo do tog koraka, koji je trebao biti najlakši, dogodilo se to da su postojala 2 postojeća rješenja za uspostavljanje veze. Ne sluteći koje je primjenjivo za konkretni slučaj, krenulo se s krivim, i tako potrošilo dosta vremena i muke u pokušavanju otklanjanja grešaka i prilagodbi, da bi se konačno ustvrdilo da to rješenje nije prikladno.

Tada se krenulo u implementaciju drugog rješenja, koje je jednostavnije, ali radi izvrsno. Pa se tu na račun mogućnosti išlo na pouzdan rad.

Kad je postignuto da sustav radi u potpunosti, moglo se početi s testiranjem, gdje su uviđeni određeni problemi i potencijalni problemi. Prije testiranja je razmotreno koji elementi sustava bi mogli loše raditi, pa su oni i testirani. Tako da će biti testirana 3 elementa

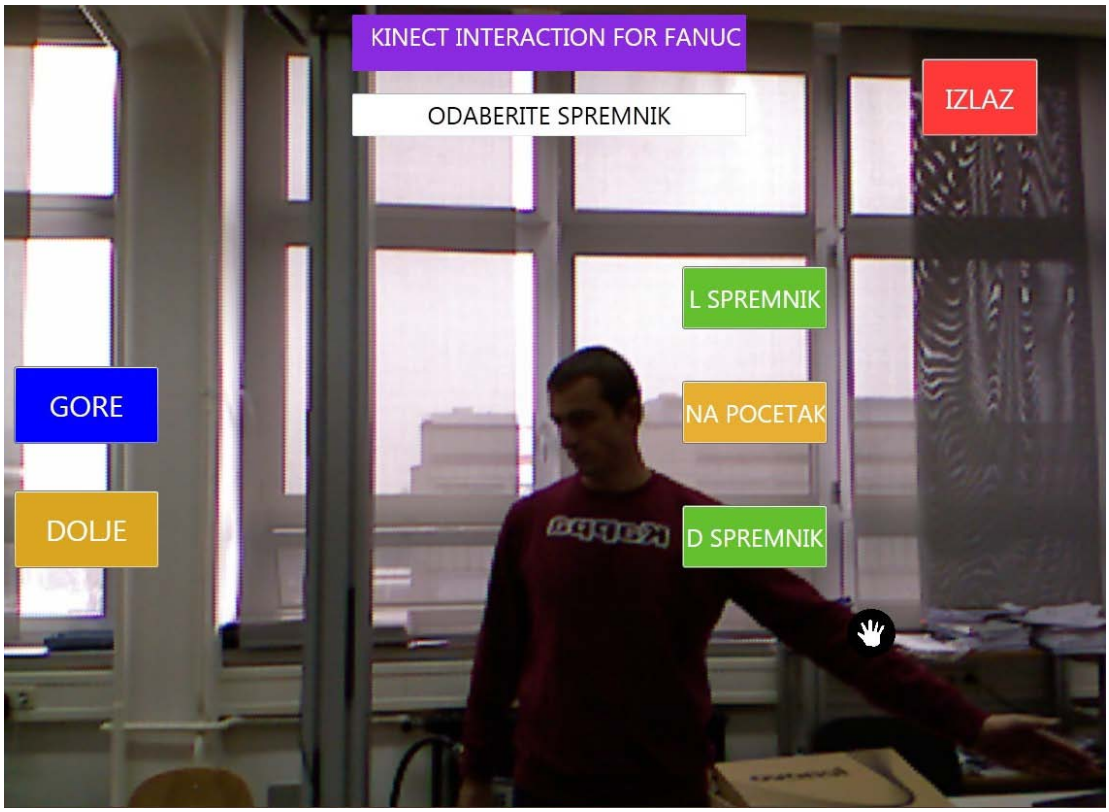
To su aplikacija, komunikacija s robotom i sam robot.

**Testiranje aplikacije**

Samo testiranje u prvoj fazi bazirano je na isprobavanju točnosti rada aplikacije, tj. njezinog dijela, praćenja šake korisnika.

Zapažene su određene nepravilnosti u rubnim dijelovima vidnog polja Kinect senzora. Te nepravilnosti su se očitovale netočnosti praćenja šake korisnika. Naime, u donjem desnom i

lijevom uglu prozora, događa se pojava da korisnik pomiče ruku , dalje u ugao, no kursor, točnije praćenje skeleta, to ne može detektirati. To je prikazano slikom (slika 44).




**Slika 44 Greška u rubnom području**

Iz tog razloga, interaktivni gumbi su postavljeni naokolo prozora , s time da se nisu postavljali u donji dio, tako da se korisnik ne treba saginjati. Iznimka je gumb za izlaz iz aplikacije koji je smješten u gornjem desnom uglu, iz razloga da se ne aktivira slučajno primjerice kad korisnik stoji u jednom položaju ne svjestan da može aktivirati gumb.

Drugi dio testiranja obuhvaća testiranje rada gumba kada su sakriveni. Naime, pošto interaktivni gumbi nisu standardni *Windows application button*, nemaju ni sve njihove značajke, nego su one izmijenjene. Konkretno, nemaju značajku isključivanja gumba. S druge strane, značajka vidljivosti gumba ima mogućnost uključivanja i isključivanja. Na slici (slika 45) se može vidjeti kako to izgleda prilikom rada aplikacije. Do ove greške dolazi jer gumbi su pozicionirani na svoje mjesto u prozoru, a pošto im je bit da se aktiviraju prilikom dolaska kursora u njihovo područje. Kada bi se ta opcija isključila u samom kodu gumba, tada bi se izgubila osnovna funkcija. Ovaj problem se može riješiti tako da svi gumbi u svakom trenutku budu prikazani na ekranu, no tada bi se izgubilo na dinamičnosti rada, i sam prozor bi bio

prenatrpan, a uostalom, bilo bi nezgrapno prikazati koji spremnik je trenutno aktivan, tako da je u konačnici odabran način rada da se gumbi aktiviraju.


**Slika 45 Greška u području gumba dok nije aktivan**

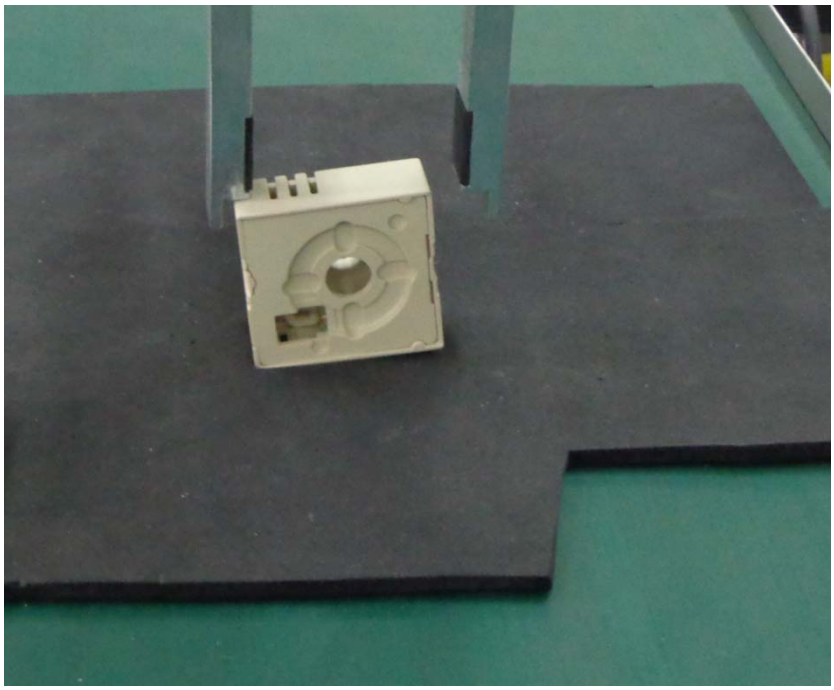
### Testiranje komunikacije

Kod testiranja komunikacije, koja je jednostavna, sustav radi izvrsno. Svaka poruka koja se šalje robotskoj upravljačkoj jedinici se uspješno pročita i pokrene adekvatni program na robotu.

### Testiranje rada robota

Zadnji dio testiranja je rad robota, točnije vizijskog sustava. Naime, prilikom izvođenja vizijskog procesa može se doći do određenih grešaka koje su posljedica osvjetljenja. Naime, može doći do pojave sjena koje za posljedicu imaju netočno lociranje rubova predmeta koje za sobom povlači netočni zahvat predmeta. Kod vizijskog procesa može doći i do preklapanja, što znači da će prilikom traženja predmeta doći do pogrešnih rubova. To će pak rezultirati krivim izračunom težišta, pa će hvataljka krivo zahvatiti predmet.

Zbog mogućih problema vizijskog sustava na radnoj površini, koji su prikazani na slici (slika 46), u principu, uvijek je dobro imati na radnoj površini dodatnu zaštitu, u vidu spužvice. Tako da u situaciji kada vizijski sustav krivo detektira težište predmeta rada i kad se hvataljka počne zabijati u predmet, točnije predmet rada, spužva to može amortizirati, i ne će doći do oštećenja robota ni predmeta rada.



**Slika 46 Opasna greška vizijskog sustava**

Ovo je dakako alternativno rješenje, a najsigurnije bi bilo da je robot opremljen senzorom sile koji bi zaustavljao gibanje dok sila na hvataljku prelazi određeni iznos.

Na slici (slika46) je namjerno prikazano zabijanje hvataljke u podlogu, kako bi se upozorilo na tu potencijalnu opasnost za sustav. Vjerojatno najbolja preventiva ovom problemu je smanjenje brzine gibanja robota koja se jednostavno može podesiti na samom *i*Pendantu i prilikom rada na diplomskom radu ona nje prelazila 50 % od maksimalnog iznosa, zbog same mogućnosti greške vizijskog sustava, ali i zbog konstrukcije okvira robotske stanice, unutar kojega robot svojim inercijskim reakcijama može ozbiljno zatresti cijelu konstrukciju.

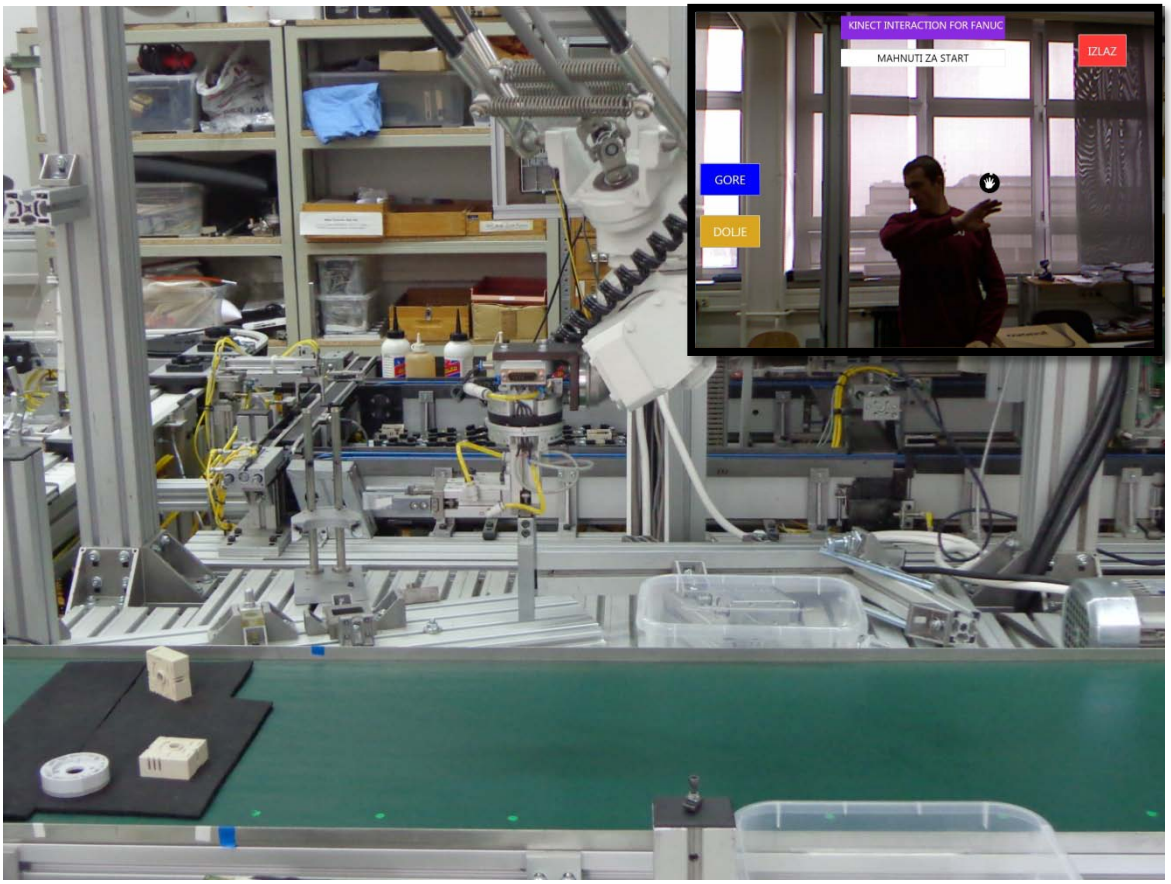
## 8.1 Opažanja u radu sustava

Cijeli sustav nakon konačnih prilagodbi radi kako je zamišljeno. Pošto je izbjegnuta upotreba kalibracije, Kinect se može premještati na bilo koju poziciju i raditi će ispravno u okviru napravljene aplikacije sve dok su zadovoljeni uvjeti osvjetljenja.

Kako je Kinect stvoren da uz sebe ima još i uređaj koji prikazuje sliku, kao što su televizor, monitor ili projektor, to uz sebe povlači činjenicu da korisnik tijekom upotrebe Kinect mora koristiti i takav uređaj, a to je u nekim slučajevima naporno, kao što je slučaj u odom radu i možda postoji način da se to zaobiđe. Bilo bi daleko jednostavnije i intuitivnije da se u okviru nekog drugog projekta pozabavim tom problematikom. Dakle, da se potpuno izbjegne korištenje sučelja na računalu ili da se koristi samo za kontrolu.

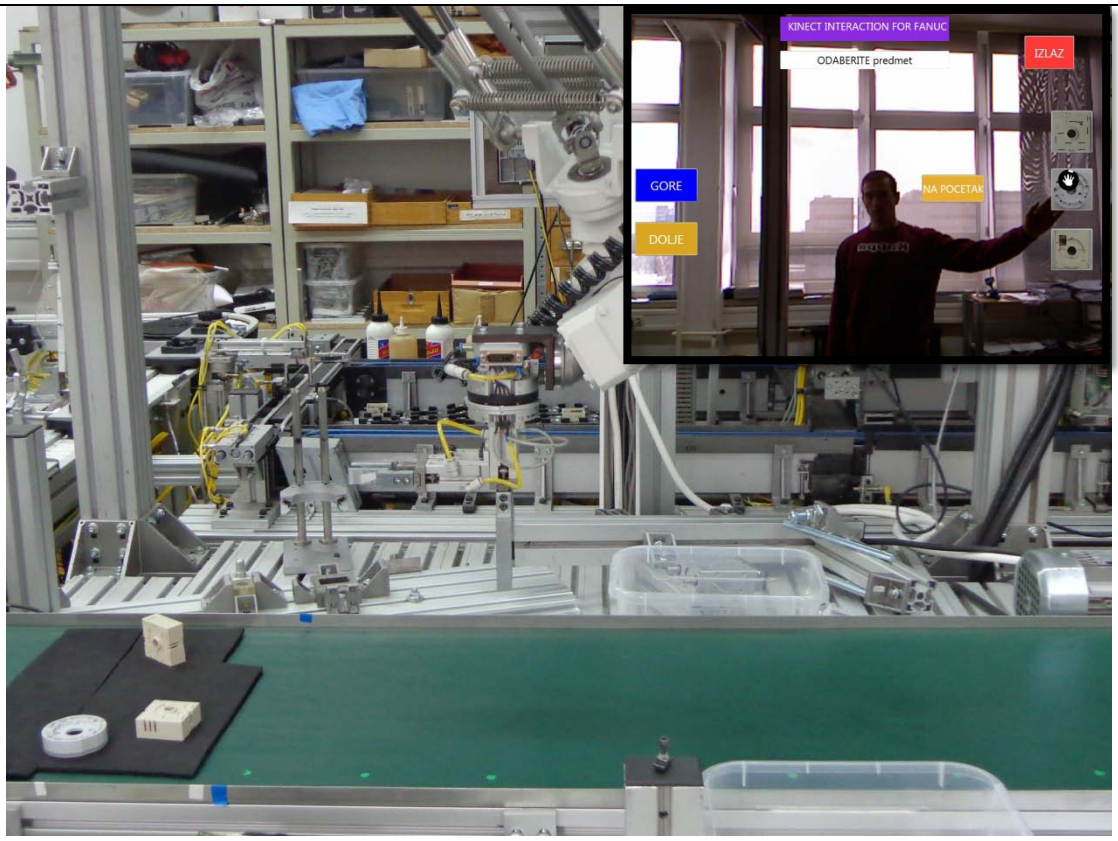
Još će biti prikazano i slijedom fotografija točno kako radi sustav od strane aplikacije i od strane robota. Na sljedećim slikama će biti prikazano u 5 koraka kako izgleda upravljanje robotom.

Prva slika (slika 47) prikazuje položaj prije aktivacije robota, gdje robot miruje, a u aplikaciji korisnik počinje mahati robotu za uspostavu komunikacije.




**Slika 47 Početno stanje**

Druga slika (slika 48) prikazuje aktiviranog robota u aplikaciji i odabir spremnika i predmeta., dok za to vrijeme robot još miruje.

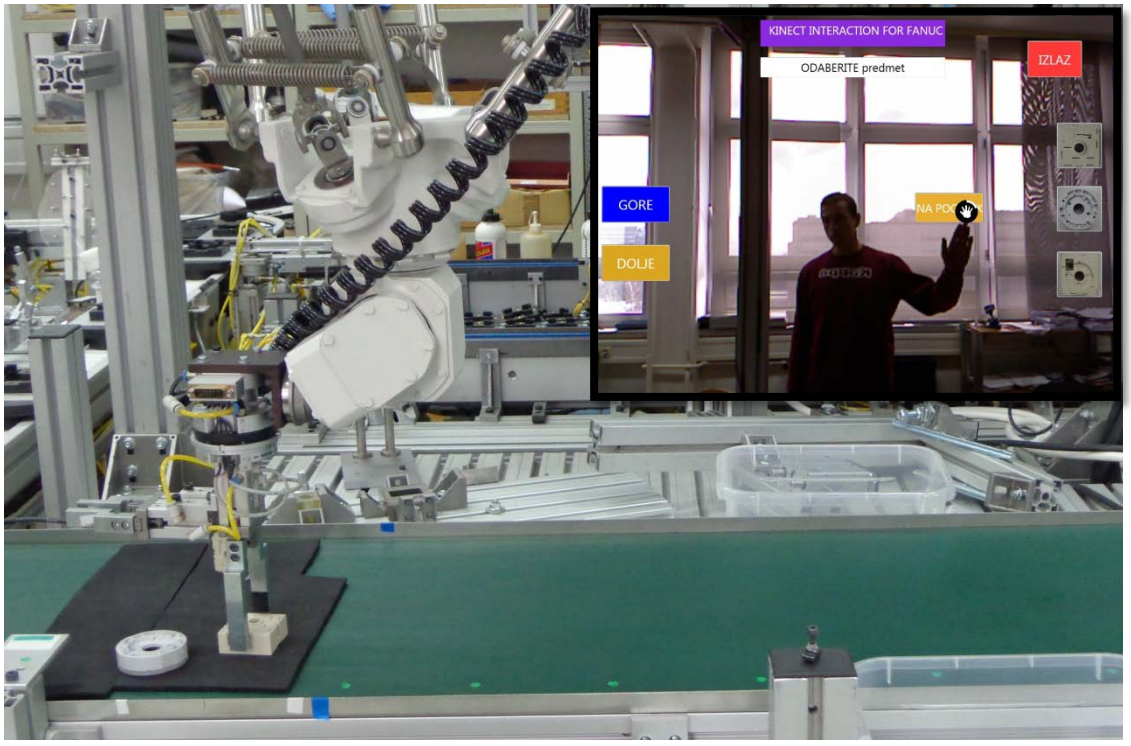



**Slika 48 Drugi korak**

Treća slika (slika 49) prikazuje trenutak prije kad robot krene zahvaćati predmet. Dakle, poruka o odabranom predmetu i spremniku je uspješno primljena. A korisnik u prozoru aplikacije miruje. Istodobno je dobro da se prati gibanje robota, i u slučaju kakvih nepredviđenih kretanja da se može pravovremeno zaustaviti gibanje.

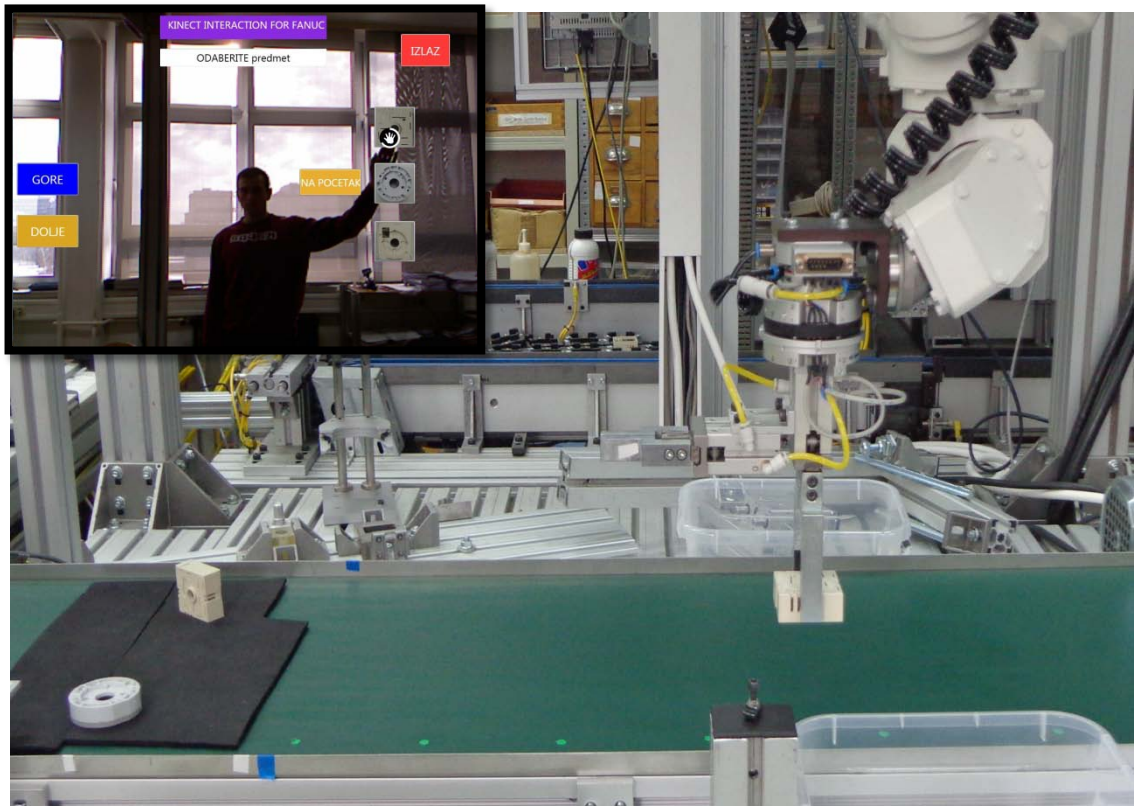

**Slika 49 Treći korak**

Četvrta slika (slika 50) prikazuje zahvat predmeta, dok aplikacija prikazuje da se korisnik za to vrijeme vraća na izbor spremnika.




**Slika 50 Četvrti korak**

Peta slika (slika 51) prikazuje kako robot odlaže predmet, a korisnik odabire drugi predmet rada.




**Slika 51 Peti korak**

Na priloženom CD-R disku nalazi se video koji prikazuje navedeni postupak tako što će se istovremeno prikazivati što vidi Kinect i samo sučelje, te što radi robot.

## 8.2 Nedostatci i prijedlog poboljšanja

Glavni nedostatak sustava je taj što nema povratnu vezu od strane robota, kako bi se spriječilo eventualno nepredviđeno gibanje. Ukoliko robot naiđe na kakvu prepreku koja se pojavi priliko izvođenje programa, on ne može nikako to primijetiti, a nakon toga ni dojaviti korisniku o tome.

Moguća poboljšanja aplikacije koja je temelj rada su raznovrsna, a mogu se izdvojiti sljedeća. Primjerice, trebalo bi dodati posebno sučelje za korištenje glasovnih naredbi i osmisliti algoritam i program koji bi omogućavao snimanje glasovne naredbe, koja bi se tada mogla pridodati nekoj novoj ili postojećoj naredbi. Dakako, kod glasovnih naredbi i glasovnog upravljanja, treba pripaziti kakve će se riječi koristiti, i ne uzimati svakodnevne, jer bi to moglo interferirati s naredbama robotu.

Drugo poboljšanje bi moglo biti korištenje pametno osmišljenih gesta za upravljanje robotom. Za tu primjenu trebalo bi osmisliti i posebni način kalibracije, pošto Kinect nije precizan za određenu vrstu gesti.

Također implementirati u sustav upravljanje nove vrste praćenja od strane Kinecta, koje su se pojavile za vrijeme izrade rada. To su primjerice praćenje lica korisnike, gdje se lice prepoznaje posebnim algoritmom koji radi svojevrsnu mrežu konačnih trokutnih elemenata koji dalje služe za procesiranje podataka. Isto tako izlaskom nove verzije Kinecta, koji će imati mogućnost čitanja s usana, to bi se moglo iskoristiti kao podloga za nove ideje upravljanja.

Još jedna mogućnost poboljšanje je korištenje 2 Kinect senzora koji bi objedinjavali i upravljanje i nadzor.

Konačno, zanimljivo bi bilo pronaći svrhovitu primjenu Kinecta u industrijskoj primjeni, ne samo u pokazne svrhe, nego primjerice i u serijskoj proizvodnji, gdje bi se Kinect mogao koristiti za kontrolu, upravljanje i slične namjene.

# 9 ZAKLJUČAK

Korištenjem stečenih znanja o radu Kinect senzora i njegovim mogućnostima, te potrebom da se to ugradi na robotski sustav, nastala je aplikacija koja je plod isprobavanja i namještanja parametara Kinect senzora. Kao takva, zaokružena je cjelina koja objedinjuje nekoliko elemenata u sustavu, no ima još puno prostora za proširenje, ili potpuno drugačiji pristup upravljanju robota, bez sučelja. Aplikacija izgleda minimalistički, a takve su joj i mogućnosti, no kako je svrha rada bila pokazati mogućnost primjene Kinecta za manipulaciju predmetima koristeći robota kao izvršnog člana, tako da se ostavi prostor za daljnje ideje iskorištenja Kinecta kao ulazne jedinice u sustav. Naglasak u ovom radu je stavljen na sami Kinect senzor, i kao takav, robot nije presudan faktor u cijelom sustavu, tj. može biti bilo koji robot koji ima komunikaciju s okolinom.

Prilikom izrade rada bilo je nekoliko prekretnica koje su odredile daljnji tijek razvoja, tako da je od početne zamisli rad u nekoliko aspekata pojednostavljen zbog ograničenja u vidu opreme i vremena istraživanja.

Nakon završetka izrade diplomskog rada i testiranja rada sustava valja zaključiti da su u dobrom dijelu odrađene prvotne zamišljene radnje koje bi sustav trebao biti u stanju izvesti.

Pojednostavili su se dijelovi u vidu komunikacije s robotom, koja je svedena je jednostranu, tako da ona dosljedno radi.

Iako u praktičnoj industrijskoj primjeni ovakav sustav ne bi bio dovoljno efikasan ni profitabilan, daljnjim razvojem uređaja sličnih Kinectu, može se dobiti na točnosti i samim time, otvorit će se nove mogućnosti za iskorištenje potencijala ovakvih sustava, pošto je sadašnje stanje, da je to još relativno novi proizvod i još traži svoju glavnu primjenu u produktivnim granama, ako se izuzme njegova primarna primjena u igraćoj industriji. Korist sustava bi se mogla primijeniti primjerice u budućim robotskim sustavima koji bi se primjenjivali u domaćinstvu, gdje je korisniku bitno da ima jednostavno sučelje, i da ne mora brinuti o procesima koji je se odvijaju u pozadini.

# Literatura

[1] Travis Pope. enConnected. [Online]. http://enconnected.com/2011/06/16/kinect-software-development-tools-for-windows-debut-on-msdn/

[2] Josh Lowenshon. C Net. [Online]. http://news.cnet.com/8301-10797_3-20022236-235.html

[3] Julian Horsey. Geeky Gadgets. [Online]. http://www.geeky-gadgets.com/ifixit-disassemble-microsofts-kinect-05-11-2010/

[4] John Biggs. Techcrunch. [Online]. http://techcrunch.com/2010/11/04/kinect-disconnected-ifixits-kinect-teardown/

[5] MSXBOX. [Online]. http://www.msxbox-world.com/xbox360/kinect/faqs/305/kinect-technical-specifications.html

[6] Optoelectronics. [Online]. http://ntuzhchen.blogspot.com/2010/12/how-kinect-works-prime-sense.html

[7] Travis Deyle. Hizook. [Online]. http://www.hizook.com/blog/2010/03/28/low-cost-depth-cameras-aka-ranging-cameras-or-rgb-d-cameras-emerge-2010

[8] Renaund Dumond. Renauddumont. [Online]. http://www.renauddumont.be/en/2012/kinect-sdk-1-0-3-tracker-les-mouvements-avec-le-skeletonstream

[9] Luiz Lopez. Profesor Luiz Lopez. [Online]. http://professorluizlopes.wordpress.com/category/kinect/

[10] Wikipedia. [Online]. http://en.wikipedia.org/wiki/Graphical_user_interface

[11] Ante Bučević, *Vođenje robota pokazivanjem*. Zagreb, 2011.

[12] Fanuc Robotics. Products. [Online]. http://www.fanucrobotics.com/Products/Robots/Atoz.aspx

[13] Sony. Camera XC 56. [Online]. http://www.pro.sony.eu/pro/lang/en/eu/product/prog/xc-56/specifications#specifications

[14] Tamron. [Online]. http://www.tamron.co.jp/en/data/cctv_ir/12vm1040asir.html

[15] Msdn. MSC United Kingdom. [Online].

http://blogs.msdn.com/b/mcsuksoldev/archive/2011/08/08/writing-a-gesture-service-with-the-kinect-for-windows-sdk.aspx

[16] Rob Miles, *Start Here Learn Microsoft Kinect API*. Redmond, SAD: Microsofr Press, 2012.

[17] James Ashley Jarret Webb, *Beginning Kinecr programming with the Microsoft Kinect SDK*.: Apress, 2011.

[18] Skupina autora, *FANUC Robotics SYSTEM R-30iA Controller KAREL Reference Manual*.: Fanuc Robotics Amrica, 2007.

[19] Bojan Šekoranja, Bojan Jerbić Marko Švaco, *Proframiranje Fanuc robota - osnove*. Zagreb, 2011.

Zadnji pristup Online materijalima, 2.2.2013.

# PRILOZI

I. Dio programskog koda aplikacije

II. Programski kod vizualnog dijela aplikacije

III. LIB_FILE.kl

IV. KIN_ORIG5.kl

V. CD-R disc sa digitalinom verzijom rada sa svim materijalima korištenima u radu.

# *I. Dio programskog koda aplikacije*

```
#region USING
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
// dodano za  Kinect
using Beginning.Kinect.Framework.Controls;
using Microsoft.Kinect;
using WaveDetection;
using Beginning.Kinect.Framework;
using Coding4Fun.Kinect.Wpf;
//dodamo za stream
using System.Net.Sockets;
using Utilities;
//timer
using System.Windows.Threading;
#endregion using

namespace KinectFrameworkTest
{
   /// <summary>
   /// Interaction logic for MainWindow.xaml
   /// </summary>
   ///

   public partial class MainWindow : Window
   {
      private KinectSensor _kinectDevice;
      private Skeleton[] _FrameSkeletons;
      private WaveGesture _WaveGesture;

      public static string start, odabir;

      //globalne za stream - Bucevic
      TcpClient tcpClient = new TcpClient();

     //varijable za Kinect button

      private KinectSensor _Kinect;
      private WriteableBitmap _ColorImageBitmap;
      private Int32Rect _ColorImageBitmapRect;
      private int _ColorImageStride;
      private Skeleton[] FrameSkeletons;


      List<Button> buttons;
      static Button selected;

      float handX;
      float handY;

      public MainWindow()
      {
```

```
        InitializeComponent();
        this._WaveGesture = new WaveGesture();
        this._WaveGesture.GestureDetected += new EventHandler(_WaveGesture_GestureDetected);

        this._kinectDevice = KinectSensor.KinectSensors.FirstOrDefault(x => x.Status == KinectStatus.Connected);
        this._kinectDevice.SkeletonFrameReady += KinectDevice_SkeletonFrameReady;

        InitializeComponent();

        InitializeButtons();
        kinectButton.Click += new RoutedEventHandler(kinectButton_Click);

        this.Loaded += (s, e) => { DiscoverKinectSensor(); };
        this.Unloaded += (s, e) => { this.Kinect = null; };

    }

    #region window loaded
    private void Window_Loaded(object sender, RoutedEventArgs e)
    {
      listBox1.Items.Add(string.Format("MAHNITE ZA START"));
      listBox1.SelectedIndex = listBox1.Items.Count - 1;
      listBox1.SelectedIndex = -1;

      button_01.IsEnabled = false;
      button_02.IsEnabled = false;
      button_1.IsEnabled = false;
      button_2.IsEnabled = false;
      button_3.IsEnabled = false;
      button_4.IsEnabled = false;
      button_5.IsEnabled = false;
      button_6.IsEnabled = false;

    }
    #endregion window loaded

    #region Skelet
    private void KinectDevice_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
    {
      using (SkeletonFrame frame = e.OpenSkeletonFrame())
      {
        if (frame != null)
        {
          this._FrameSkeletons = new Skeleton[_kinectDevice.SkeletonStream.FrameSkeletonArrayLength];
          frame.CopySkeletonDataTo(this._FrameSkeletons);

          DateTime startMarker = DateTime.Now;
          this._WaveGesture.Update(this._FrameSkeletons, frame.Timestamp);
        }
      }
    }
    #endregion Skelet

    #region Prepoznavanje gesti

    private void _WaveGesture_GestureDetected(object sender, EventArgs e)
    {
      while (pom1 == true)
      {
        listBox1.Items.Add(string.Format("STARTANO - POKRENUTA VEZA u {0}",
DateTime.Now.ToLongTimeString()));
        listBox1.SelectedIndex = listBox1.Items.Count - 1;
        listBox1.SelectedIndex = -1;
        listBox1.Items.Add(string.Format("ODABERITE SPREMNIK"));
        listBox1.SelectedIndex = listBox1.Items.Count - 1;
        listBox1.SelectedIndex = -1;
```

```
            txt_box.Text = "ODABERITE SPREMNIK";

            //za startanje veze se robotom

            tcpClient.Connect("192.168.123.27", 5555);

            /////// stream dio
            NetworkStream NS = tcpClient.GetStream();
            if (tcpClient.Connected && NS.CanWrite)
            {

               {
                  Byte[] podatakmem = Encoding.ASCII.GetBytes(startanje);
                  NS.Write(podatakmem, 0, podatakmem.Length);

               }
            }
            pom1 = false;
         }
         // startanje gumbova
         button_01.Visibility = Visibility.Visible;
         button_02.Visibility = Visibility.Visible;
         button_N.Visibility = Visibility.Visible;
         button_01.IsEnabled = IsEnabled;
         button_02.IsEnabled = IsEnabled;
         button_N.IsEnabled = IsEnabled;

      }

      private void _UniversalPause_PauseDetected(object sender, EventArgs e)
      {

      }
      #endregion Preoznavanje gesti

      #region CFF Kinect Button

      //initialize buttons to be checked
      private void InitializeButtons()
      {
         buttons = new List<Button> { quitButton, button_1, button_2, button_3, button_4, button_5, button_6, button_N,
button_01, button_02, button_UP, button_DOWN };
      }

      //raise event for Kinect sensor status changed
      private void DiscoverKinectSensor()
      {
         KinectSensor.KinectSensors.StatusChanged += KinectSensors_StatusChanged;
         this.Kinect = KinectSensor.KinectSensors.FirstOrDefault(x => x.Status == KinectStatus.Connected);
      }

      private void KinectSensors_StatusChanged(object sender, StatusChangedEventArgs e)
      {
         switch (e.Status)
         {
            case KinectStatus.Connected:
               if (this.Kinect == null)
               {
                  this.Kinect = e.Sensor;
               }
               break;
            case KinectStatus.Disconnected:
               if (this.Kinect == e.Sensor)
               {
                  this.Kinect = null;
```

```
                this.Kinect = KinectSensor.KinectSensors.FirstOrDefault(x => x.Status == KinectStatus.Connected);
                if (this.Kinect == null)
                {
                    MessageBox.Show("Sensor Disconnected. Please reconnect to continue.");
                }
            }
            break;
    }
}

public KinectSensor Kinect
{
    get { return this._Kinect; }
    set
    {
        if (this._Kinect != value)
        {
            if (this._Kinect != null)
            {
                UninitializeKinectSensor(this._Kinect);
                this._Kinect = null;
            }
            if (value != null && value.Status == KinectStatus.Connected)
            {
                this._Kinect = value;
                InitializeKinectSensor(this._Kinect);
            }
        }
    }
}

private void UninitializeKinectSensor(KinectSensor kinectSensor)
{
    if (kinectSensor != null)
    {
        kinectSensor.Stop();
        kinectSensor.ColorFrameReady -= Kinect_ColorFrameReady;
        kinectSensor.SkeletonFrameReady -= Kinect_SkeletonFrameReady;
    }
}

private void InitializeKinectSensor(KinectSensor kinectSensor)
{
    if (kinectSensor != null)
    {
        ColorImageStream colorStream = kinectSensor.ColorStream;
        colorStream.Enable();
        this._ColorImageBitmap = new WriteableBitmap(colorStream.FrameWidth, colorStream.FrameHeight,
            96, 96, PixelFormats.Bgr32, null);
        this._ColorImageBitmapRect = new Int32Rect(0, 0, colorStream.FrameWidth, colorStream.FrameHeight);
        this._ColorImageStride = colorStream.FrameWidth * colorStream.FrameBytesPerPixel;
        videoStream.Source = this._ColorImageBitmap;

        kinectSensor.SkeletonStream.Enable(new TransformSmoothParameters()
        {
            Correction = 0.5f,
            JitterRadius = 0.05f,
            MaxDeviationRadius = 0.04f,
            Smoothing = 0.5f
        });

        kinectSensor.SkeletonFrameReady += Kinect_SkeletonFrameReady;
        kinectSensor.ColorFrameReady += Kinect_ColorFrameReady;
        kinectSensor.Start();
        this.FrameSkeletons = new Skeleton[this.Kinect.SkeletonStream.FrameSkeletonArrayLength];
```

```
        }
    }

    private void Kinect_ColorFrameReady(object sender, ColorImageFrameReadyEventArgs e)
    {
        using (ColorImageFrame frame = e.OpenColorImageFrame())
        {
            if (frame != null)
            {
                byte[] pixelData = new byte[frame.PixelDataLength];
                frame.CopyPixelDataTo(pixelData);
                this._ColorImageBitmap.WritePixels(this._ColorImageBitmapRect, pixelData,
                    this._ColorImageStride, 0);
            }
        }
    }

    private void Kinect_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
    {
        using (SkeletonFrame frame = e.OpenSkeletonFrame())
        {
            if (frame != null)
            {
                frame.CopySkeletonDataTo(this.FrameSkeletons);
                Skeleton skeleton = GetPrimarySkeleton(this.FrameSkeletons);

                if (skeleton == null)
                {
                    kinectButton.Visibility = Visibility.Collapsed;
                }
                else
                {
                    Joint primaryHand = GetPrimaryHand(skeleton);
                    TrackHand(primaryHand);

                }
            }
        }
    }

    //track and display hand
    private void TrackHand(Joint hand)
    {
        if (hand.TrackingState == JointTrackingState.NotTracked)
        {
            kinectButton.Visibility = System.Windows.Visibility.Collapsed;
        }
        else
        {
            kinectButton.Visibility = System.Windows.Visibility.Visible;

            DepthImagePoint point = this.Kinect.MapSkeletonPointToDepth(hand.Position,
DepthImageFormat.Resolution640x480Fps30);
            handX = (int)((point.X * LayoutRoot.ActualWidth / this.Kinect.DepthStream.FrameWidth) -
                (kinectButton.ActualWidth / 2.0));
            handY = (int)((point.Y * LayoutRoot.ActualHeight / this.Kinect.DepthStream.FrameHeight) -
                (kinectButton.ActualHeight / 2.0));
            Canvas.SetLeft(kinectButton, handX);
            Canvas.SetTop(kinectButton, handY);

            if (isHandOver(kinectButton, buttons)) kinectButton.Hovering();
            else kinectButton.Release();
            if (hand.JointType == JointType.HandRight)
            {
                kinectButton.ImageSource = "/Images/RightHand.png";
                kinectButton.ActiveImageSource = "/Images/RightHand.png";
```

```
            }
            else
            {
                kinectButton.ImageSource = "/Images/LeftHand.png";
                kinectButton.ActiveImageSource = "/Images/LeftHand.png";
            }
        }
    }

    //detect if hand is overlapping over any button
    private bool isHandOver(FrameworkElement hand, List<Button> buttonslist)
    {
        var handTopLeft = new Point(Canvas.GetLeft(hand), Canvas.GetTop(hand));
        var handX = handTopLeft.X + hand.ActualWidth / 2;
        var handY = handTopLeft.Y + hand.ActualHeight / 2;
        foreach (Button target in buttonslist)
        {
            Point targetTopLeft = new Point(Canvas.GetLeft(target), Canvas.GetTop(target));
            if (handX > targetTopLeft.X &&
                handX < targetTopLeft.X + target.Width &&
                handY > targetTopLeft.Y &&
                handY < targetTopLeft.Y + target.Height)
            {
                selected = target;
                return true;
            }
        }
        return false;
    }

    //get the hand closest to the Kinect sensor
    private static Joint GetPrimaryHand(Skeleton skeleton)
    {
        Joint primaryHand = new Joint();
        if (skeleton != null)
        {
            primaryHand = skeleton.Joints[JointType.HandLeft];
            Joint rightHand = skeleton.Joints[JointType.HandRight];
            if (rightHand.TrackingState != JointTrackingState.NotTracked)
            {
                if (primaryHand.TrackingState == JointTrackingState.NotTracked)
                {
                    primaryHand = rightHand;
                }
                else
                {
                    if (primaryHand.Position.Z > rightHand.Position.Z)
                    {
                        primaryHand = rightHand;
                    }
                }
            }
        }
        return primaryHand;
    }

    //get the skeleton closest to the Kinect sensor
    private static Skeleton GetPrimarySkeleton(Skeleton[] skeletons)
    {
        Skeleton skeleton = null;
        if (skeletons != null)
        {
            for (int i = 0; i < skeletons.Length; i++)
            {
                if (skeletons[i].TrackingState == SkeletonTrackingState.Tracked)
                {
```

```
                if (skeleton == null)
                {
                    skeleton = skeletons[i];
                }
                else
                {
                    if (skeleton.Position.Z > skeletons[i].Position.Z)
                    {
                        skeleton = skeletons[i];
                    }
                }
            }
        }
    }
    return skeleton;
}

void kinectButton_Click(object sender, RoutedEventArgs e)
{
    selected.RaiseEvent(new RoutedEventArgs(Button.ClickEvent, selected));
}


private void quitButton_Click(object sender, RoutedEventArgs e)
{
    Application.Current.Shutdown();
}

#endregion CFF Kinect Button

#region Hover Gumbi

// pom varijable za slanje poruka
string startanje =  "0123";
string pozivanje1 = "A123";
string pozivanje2 = "B123";
string pozivanje3 = "C123";
string pozivanje4 = "D123";
string pozivanje5 = "E123";
string pozivanje6 = "F123";
string stop = "S123";

// pom var za petlje
bool pom1 = true;

private void button_1_Click(object sender, RoutedEventArgs e)
{
    ///// stream dio
    NetworkStream NS = tcpClient.GetStream();
    if (tcpClient.Connected && NS.CanWrite)
    {
        Byte[] salji2 = Encoding.ASCII.GetBytes(pozivanje1);
        NS.Write(salji2, 0, salji2.Length);

    }
}

private void button_2_Click(object sender, RoutedEventArgs e)
{

    ///// stream dio
    NetworkStream NS = tcpClient.GetStream();
    if (tcpClient.Connected && NS.CanWrite)
    {
        Byte[] salji3 = Encoding.ASCII.GetBytes(pozivanje2);
        NS.Write(salji3, 0, salji3.Length);
```

```
        // txtspremi.Text += "\n" + txtspremi.Text;

    }
}

private void button_3_Click(object sender, RoutedEventArgs e)
{

    ///// stream dio
    NetworkStream NS = tcpClient.GetStream();
    if (tcpClient.Connected && NS.CanWrite)
    {
        Byte[] salji3 = Encoding.ASCII.GetBytes(pozivanje3);
        NS.Write(salji3, 0, salji3.Length);

        // txtspremi.Text += "\n" + txtspremi.Text;

    }
}

private void button_4_Click(object sender, RoutedEventArgs e)
{
    ///// stream dio
    NetworkStream NS = tcpClient.GetStream();
    if (tcpClient.Connected && NS.CanWrite)
    {
        Byte[] salji4 = Encoding.ASCII.GetBytes(pozivanje4);
        NS.Write(salji4, 0, salji4.Length);

    }
}

private void button_5_Click(object sender, RoutedEventArgs e)
{
    ///// stream dio
    NetworkStream NS = tcpClient.GetStream();
    if (tcpClient.Connected && NS.CanWrite)
    {
        Byte[] salji5 = Encoding.ASCII.GetBytes(pozivanje5);
        NS.Write(salji5, 0, salji5.Length);

    }
}

private void button_6_Click(object sender, RoutedEventArgs e)
{
        ///// stream dio
    NetworkStream NS = tcpClient.GetStream();
    if (tcpClient.Connected && NS.CanWrite)
    {
        Byte[] salji6 = Encoding.ASCII.GetBytes(pozivanje6);
        NS.Write(salji6, 0, salji6.Length);

    }
}

private void Stop_Click(object sender, RoutedEventArgs e)
{
    ///// stream dio
    NetworkStream NS = tcpClient.GetStream();
    if (tcpClient.Connected && NS.CanWrite)
    {
        Byte[] podatakmem = Encoding.ASCII.GetBytes(stop);
        NS.Write(podatakmem, 0, podatakmem.Length);
```

```
            // txtspremi.Text += "\n" + txtspremi.Text;
        }
    }

    private void button_01_Click(object sender, RoutedEventArgs e)
    {
        button_1.Visibility = Visibility.Visible;
        button_2.Visibility = Visibility.Visible;
        button_3.Visibility = Visibility.Visible;
        button_01.Visibility = Visibility.Hidden;
        button_02.Visibility = Visibility.Hidden;

        button_1.IsEnabled = true;
        button_2.IsEnabled = true;
        button_3.IsEnabled = true;
        button_01.IsEnabled = false;
        button_02.IsEnabled = false;

        listBox1.Items.Add(string.Format("ODABERITE PREDMET RADA"));
        listBox1.SelectedIndex = listBox1.Items.Count - 1;
        listBox1.SelectedIndex = -1;

        txt_box.Text = "ODABERITE predmet";
    }

    private void button_02_Click(object sender, RoutedEventArgs e)
    {
        button_4.Visibility = Visibility.Visible;
        button_5.Visibility = Visibility.Visible;
        button_6.Visibility = Visibility.Visible;
        button_01.Visibility = Visibility.Hidden;
        button_02.Visibility = Visibility.Hidden;

        button_4.IsEnabled = true;
        button_5.IsEnabled = true;
        button_6.IsEnabled = true;
        button_01.IsEnabled = false;
        button_02.IsEnabled = false;

        listBox1.Items.Add(string.Format("ODABERITE PREDMET RADA"));
        listBox1.SelectedIndex = listBox1.Items.Count - 1;
        listBox1.SelectedIndex = -1;

        txt_box.Text = "ODABERITE predmet";
    }

    private void button_n_Click(object sender, RoutedEventArgs e)
    {
        button_1.Visibility = Visibility.Hidden;
        button_2.Visibility = Visibility.Hidden;
        button_3.Visibility = Visibility.Hidden;
        button_4.Visibility = Visibility.Hidden;
        button_5.Visibility = Visibility.Hidden;
        button_6.Visibility = Visibility.Hidden;
        button_01.Visibility = Visibility.Visible;
        button_02.Visibility = Visibility.Visible;
        button_1.IsEnabled = false;
        button_2.IsEnabled = false;
        button_3.IsEnabled = false;
        button_4.IsEnabled = false;
        button_5.IsEnabled = false;
        button_6.IsEnabled = false;
        button_01.IsEnabled = true;
        button_02.IsEnabled = true;
        txt_box.Text = "ODABERITE SPREMNIK";
```

```
        }

        private void up_Click(object sender, RoutedEventArgs e)
        {
            _kinectDevice.ElevationAngle += 10;
        }

        private void down_Click(object sender, RoutedEventArgs e)
        {
            _kinectDevice.ElevationAngle -= 10;
        }

        #endregion Hover Gumbi
    }
}
```

## Dodatak – datoteka *vawe_gasture.cs* za prepoznavanje geste mahanja

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Kinect;

namespace WaveDetection
{
    public class WaveGesture
    {
        #region Member Variables
        private const float WAVE_THRESHOLD          = 0.1f;
        private const int WAVE_MOVEMENT_TIMEOUT     = 5000;
        private const int LEFT_HAND                 = 0;
        private const int RIGHT_HAND                = 1;
        private const int REQUIRED_ITERATIONS       = 3;

        private WaveGestureTracker[,] _PlayerWaveTracker = new WaveGestureTracker[6,2];

        public event EventHandler GestureDetected;
        #endregion Member Variables


        #region Methods
        public void Update(Skeleton[] skeletons, long frameTimestamp)
        {
            if(skeletons != null)
            {
                Skeleton skeleton;
                for(int i = 0; i < skeletons.Length; i++)
                {
                    skeleton = skeletons[i];

                    if(skeleton.TrackingState != SkeletonTrackingState.NotTracked)
                    {
                        TrackWave(skeleton, true, ref this._PlayerWaveTracker[i, LEFT_HAND], frameTimestamp);
                        TrackWave(skeleton, false, ref this._PlayerWaveTracker[i, RIGHT_HAND], frameTimestamp);
                    }
                    else
                    {
                        this._PlayerWaveTracker[i, LEFT_HAND].Reset();
                        this._PlayerWaveTracker[i, RIGHT_HAND].Reset();
                    }
                }
            }
```

```
    }

    private void TrackWave(Skeleton skeleton, bool isLeft, ref WaveGestureTracker tracker, long timestamp)
    {
        JointType handJointId       = (isLeft) ? JointType.HandLeft : JointType.HandRight;
        JointType elbowJointId      = (isLeft) ? JointType.ElbowLeft : JointType.ElbowRight;
        Joint hand                  = skeleton.Joints[handJointId];
        Joint elbow                 = skeleton.Joints[elbowJointId];

        if(hand.TrackingState != JointTrackingState.NotTracked && elbow.TrackingState !=
JointTrackingState.NotTracked)
        {
            if(tracker.State == WaveGestureState.InProgress && tracker.Timestamp + WAVE_MOVEMENT_TIMEOUT <
timestamp)
            {
                tracker.UpdateState(WaveGestureState.Failure, timestamp);
                System.Diagnostics.Debug.WriteLine("Fail!");
            }
            else if(hand.Position.Y > elbow.Position.Y)
            {
                //Using the raw values where (0, 0) is the middle of the screen.  From the user's perspective, the X-Axis grows
more negative left and more positive right.
                if(hand.Position.X <= elbow.Position.X - WAVE_THRESHOLD)
                {
                    tracker.UpdatePosition(WavePosition.Left, timestamp);
                }
                else if(hand.Position.X >= elbow.Position.X + WAVE_THRESHOLD)
                {
                    tracker.UpdatePosition(WavePosition.Right, timestamp);
                }
                else
                {
                    tracker.UpdatePosition(WavePosition.Neutral, timestamp);
                }

                if(tracker.State != WaveGestureState.Success && tracker.IterationCount == REQUIRED_ITERATIONS)
                {
                    tracker.UpdateState(WaveGestureState.Success, timestamp);
                    System.Diagnostics.Debug.WriteLine("Success!");

                    if(GestureDetected != null)
                    {
                        GestureDetected(this, new EventArgs());
                    }
                }
            }
            else
            {
                if(tracker.State == WaveGestureState.InProgress)
                {
                    tracker.UpdateState(WaveGestureState.Failure, timestamp);
                    System.Diagnostics.Debug.WriteLine("Fail!");
                }
                else
                {
                    tracker.Reset();
                }
            }
        }
        else
        {
            tracker.Reset();
        }
    }
    #endregion Methods
```

```
    #region Helper Objects
    private enum WavePosition
    {
       None    = 0,
       Left    = 1,
       Right   = 2,
       Neutral = 3
    }

    private enum WaveGestureState
    {
       None       = 0,
       Success    = 1,
       Failure    = 2,
       InProgress = 3
    }

    private struct WaveGestureTracker
    {
       public int IterationCount;
       public WaveGestureState State;
       public long Timestamp;
       public WavePosition StartPosition;
       public WavePosition CurrentPosition;

       public void UpdateState(WaveGestureState state, long timestamp)
       {
          State      = state;
          Timestamp  = timestamp;
       }

       public void Reset()
       {
          IterationCount     = 0;
          State              = WaveGestureState.None;
          Timestamp          = 0;
          StartPosition      = WavePosition.None;
          CurrentPosition    = WavePosition.None;
       }

       public void UpdatePosition(WavePosition position, long timestamp)
       {
          if(CurrentPosition != position)
          {
             if(position == WavePosition.Left || position == WavePosition.Right)
             {
                if(State != WaveGestureState.InProgress)
                {
                   State          = WaveGestureState.InProgress;
                   IterationCount = 0;
                   StartPosition  = position;
                }
                IterationCount++;
             }
             CurrentPosition = position;
             Timestamp       = timestamp;
          }
       }
    }
    #endregion Helper Objects
  }
}
```

## II. Programski kod vizualnog dijela aplikacije

```
<Window x:Class="KinectFrameworkTest.MainWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Title="Microsoft Kinect Interaction for Fanuc" Height="864" Width="1152"
      xmlns:my="clr-namespace:Beginning.Kinect.Framework.Controls;assembly=Beginning.Kinect.Framework"
      xmlns:Controls="clr-namespace:Coding4Fun.Kinect.Wpf.Controls;assembly=Coding4Fun.Kinect.Wpf"
      Background="#FFAD4747"
      ResizeMode="NoResize"
      Loaded="Window_Loaded">

   <Grid x:Name="LayoutRoot">
      <Image Height="864" Width="1152" HorizontalAlignment="Left" Margin="0,0,0,0" Name="videoStream"
Stretch="Fill" VerticalAlignment="Top" />
      <Canvas Background="Transparent">
         <Controls:HoverButton Margin="0" Padding="0" x:Name="kinectButton" ImageSize="50"
                        ImageSource="/Images/RightHand.png"
                        ActiveImageSource="/Images/RightHand.png"
                        TimeInterval="2000"  Panel.ZIndex="1000" Canvas.Left="0" Canvas.Top="0" />
         <Label Canvas.Left="359" Canvas.Top="10" Content="  KINECT INTERACTION FOR FANUC" Height="58"
Name="label1" FontSize="25" Width="406" Foreground="White" Background="BlueViolet" UseLayoutRounding="False"
/>
         <Button Canvas.Left="946" Canvas.Top="55" Content="IZLAZ" Height="80" Name="quitButton" Width="120"
FontSize="28" Background="#FFFF3838" Foreground="White" Click="quitButton_Click" />
         <ListBox Name="listBox1" Height="62" Canvas.Left="317" Canvas.Top="73" Width="484"
BorderThickness="1,0,1,1" FontSize="20" AllowDrop="True" TextOptions.TextHintingMode="Animated"
TextOptions.TextFormattingMode="Display" SelectionMode="Extended" Visibility="Hidden"/>
         <Button x:Name="button_01" Content="L SPREMNIK" Height="65" Canvas.Left="699" Canvas.Top="269"
Width="150" Background="#FF64C02F" FontSize="25" Click="button_01_Click" BorderThickness="4"
Foreground="White" IsHitTestVisible="False" Visibility="Hidden" />
         <Button x:Name="button_02" Content="D SPREMNIK" Height="65" Canvas.Left="699" Canvas.Top="515"
Width="150" Background="#FF64C02F" FontSize="25" Click="button_02_Click" BorderThickness="4"
Foreground="White" IsHitTestVisible="False" Visibility="Hidden" />
         <Button x:Name="button_1"  Content="" Height="100" Canvas.Left="881" Canvas.Top="234" Width="100"
FontSize="25" Click="button_1_Click" BorderThickness="4" Foreground="White" Visibility="Hidden" >
            <Button.Background>
               <ImageBrush ImageSource="Images/KONEKTOR_zkuciste.png"/>
            </Button.Background>
         </Button>
         <Button x:Name="button_2" Content="" Height="100" Canvas.Left="881" Canvas.Top="372" Width="100"
FontSize="30" Click="button_2_Click" BorderThickness="4" Foreground="White" Visibility="Hidden" >
            <Button.Background>
               <ImageBrush ImageSource="Images/krug.png"/>
            </Button.Background>
         </Button>
         <Button x:Name="button_3" Content="" Height="100" Canvas.Left="881" Canvas.Top="515" Width="100"
FontSize="30" Click="button_3_Click" BorderThickness="4" Foreground="White" Visibility="Hidden" >
            <Button.Background>
               <ImageBrush ImageSource="Images/KONEKTOR_poklopac.png"/>
            </Button.Background>
         </Button>
         <Button x:Name="button_4" Content="" Height="100" Canvas.Left="1010" Canvas.Top="234" Width="100"
FontSize="30" Click="button_4_Click" BorderThickness="4" Foreground="White" IsHitTestVisible="False"
Visibility="Hidden" >
            <Button.Background>
               <ImageBrush ImageSource="Images/KONEKTOR_zkuciste.png"/>
            </Button.Background>
         </Button>
         <Button x:Name="button_5" Content="" Height="100" Canvas.Left="1010" Canvas.Top="372" Width="100"
FontSize="30" Click="button_5_Click" BorderThickness="4" Foreground="White" IsHitTestVisible="False"
Visibility="Hidden" >
            <Button.Background>
               <ImageBrush ImageSource="Images/krug.png"/>
```

```
            </Button.Background>
        </Button>
        <Button x:Name="button_6" Content="" Height="100" Canvas.Left="1010" Canvas.Top="515" Width="100"
FontSize="30" Click="button_6_Click" BorderThickness="4" Foreground="White" IsHitTestVisible="False"
Visibility="Hidden" >
            <Button.Background>
                <ImageBrush ImageSource="Images/KONEKTOR_poklopac.png"/>
            </Button.Background>
        </Button>
        <Button x:Name="button_N" Content="NA POCETAK" Height="65" Canvas.Left="699" Canvas.Top="387"
Width="150" FontSize="25" Click="button_n_Click" BorderThickness="4" Foreground="White" IsHitTestVisible="False"
RenderTransformOrigin="0.513,0.223" Focusable="False" Visibility="Hidden" >
            <Button.Background>
                <RadialGradientBrush RadiusY="0">
                    <GradientStop Color="Black" Offset="0"/>
                    <GradientStop Color="#FFE8AE31" Offset="1"/>
                </RadialGradientBrush>
            </Button.Background>
        </Button>

        <Button x:Name="button_UP" Content="GORE" Height="80" Canvas.Left="10" Canvas.Top="372" Width="150"
Background="Blue" FontSize="30" Click="up_Click" BorderThickness="4" Foreground="White" />
        <Button x:Name="button_DOWN" Content="DOLJE" Height="80" Canvas.Left="10" Canvas.Top="500"
Width="150" Background="Goldenrod" FontSize="30" Click="down_Click" BorderThickness="4" Foreground="White" />
        <TextBox Name="txt_box" Height="44" Canvas.Left="359" TextWrapping="Wrap" Text="MAHNUTI ZA START"
Canvas.Top="91" Width="406" ScrollViewer.CanContentScroll="True" TextChanged="TextBox_TextChanged_1"
VerticalContentAlignment="Center" HorizontalContentAlignment="Center" FontSize="25" />
    </Canvas>
  </Grid>
</Window>
```

## III. Datoteka LIB_FILE

```
PROGRAM LIB_FILE
%NOLOCKGROUP
%NOPAUSE = ERROR + COMMAND + TPENABLE
--*****************************VARIJABLE*********************************
VAR
STATUS,entry:INTEGER
--***************************************************************************
--************************  RUTINE  ******************************
--***************************************************************************
ROUTINE ROBOT_(ID_:STRING; HOSTNAME_:STRING): ARRAY OF INTEGER FROM LIB_BASE

 ROUTINE OPEN_FILE_(FILE_ : FILE; TAG_ : STRING)
  VAR
    CLIENT:BOOLEAN; SC:STRING[1]; i:INTEGER
  BEGIN
   CLIENT=FALSE; SC = SUB_STR(TAG_, 1, 1);IF SC='C' THEN; CLIENT=TRUE; ENDIF

   CONNECT_:::
   CLR_IO_STAT(FILE_)
   MSG_DISCO(TAG_,STATUS); WRITE('CONNECTING.... ',TAG_,CR);
   MSG_CONNECT(TAG_,STATUS)
   OPEN FILE FILE_('rw',TAG_); STATUS = IO_STATUS(FILE_)

   IF STATUS<>0 THEN
    FOR i=1 TO 3 DO
     WRITE(CHR(128),CR); WRITE('RECONNECTING AT...',3-i,CR); DELAY 10
    ENDFOR
    GOTO CONNECT_
   ENDIF

   WRITE('Status open file:',STATUS,' TAG:',TAG_,CR)
  END OPEN_FILE_

 --***************************************************************************
 ROUTINE CLOSE_FILE_(FILE_ : FILE; TAG_ : STRING)
  BEGIN
   CLOSE FILE FILE_ ;  WRITE('Disconnecting..',CR)
   MSG_DISCO(TAG_,STATUS); WRITE(TAG_,'status disco:',STATUS,CR)
  END CLOSE_FILE_

 --***************************************************************************
```

```
-- ID:  OZNAKA ROBOTA SA KOJIM SE ZELI KOMUNICIRATI: >R1< . >R2< . >R3<
-- TIP: oznaka koja oznacava da li se podaci salju ili primaju: >S< . >C<
-- SERVER je onaj koji prvi otvarat vezu tj. daje DO i ceka na DI
-- CLIENT je onaj koji prvo ceka na servera sa wait DI, i onda daje izlaz DO da je primio
-- konekciju od SERVERA
ROUTINE HANDSHAKING_ (ID_ : STRING; TIP_ : STRING)
 VAR
  NIZ : ARRAY[10] OF INTEGER

 BEGIN

  --DOHVAT PODATAKA O ID-U TJ. ROBOTU SA KOJIM SE ZELI NAPRAVITI HANDSHAKING
  NIZ = ROBOT_(ID_,'')

  --SERVER:
  IF TIP_ = 'S'  THEN
   WRITE('DOUT',NIZ[2],'=ON',CR)
   DOUT[ NIZ[2] ] = ON
   WRITE('SERVER WAIT FOR DIN:',NIZ[2],CR)

   WHILE DIN[ NIZ[1] ] = OFF DO
    DOUT[ NIZ[2] ] = ON
    DELAY 100
   ENDWHILE

   --WAIT FOR DIN[ NIZ[1] ] = ON  --NE RADI JER SE DO UGASI NA R2 SAM OD SEBE NAKON 5 SEKUNDI!!!!
   DELAY 150
   DOUT[ NIZ[2] ] = OFF
  ENDIF

  --CLIENT:
  IF TIP_ = 'C' THEN
   WRITE('CLIENT WAIT FOR DIN:',NIZ[1],CR)
   WAIT FOR DIN[ NIZ[1] ] = ON
   DOUT[ NIZ[2] ] = ON
   DELAY 150
   DOUT[ NIZ[2] ] = OFF
  ENDIF

 END HANDSHAKING_

--*********************************************************************
ROUTINE WRITE_(STRING_ : STRING)
 BEGIN
```

```
    WRITE(STRING_,CR);
  END WRITE_

END LIB_FILE
```

## *IV. Datoteka KIN_ORIG5.kl – program Kinect_10*

```
PROGRAM Kinect_10
VAR
STATUS,entry,port: INTEGER
file_var, file_citaj : FILE
S, reply, message,start : STRING[128]
n_bytes, REG, g : INTEGER
 config_var:CONFIG

 ----------------------VANJSKE RUTINE-------------------------
 ROUTINE OPEN_FILE_(FILE_ : FILE; TAG_ : STRING) FROM LIB_FILE
 ROUTINE CLOSE_FILE_(FILE_ : FILE; TAG_ : STRING) FROM LIB_FILE
 ---------------------------------------------------------------

BEGIN
        $GROUP[1].$UFRAME = $MNUFRAME[1,1];
        $GROUP[1].$UTOOL =  $MNUTOOL[1,1]
        $GROUP[1].$MOTYPE=LINEAR;
        $GROUP[1].$SPEED=300
        $GROUP[1].$TERMTYPE=NODECEL
        port=5555
        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$OPER',0,STATUS) ;
        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$STATE',0,STATUS) ; DELAY 20
        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$COMMENT','SOUND',STATUS) ;
        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$PROTOCOL','SM',STATUS) ;
         SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$REPERRS','FALSE',STATUS) ;
        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$TIMEOUT',9999,STATUS) ;
        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$PWRD_TIMEOUT',0,STATUS) ;
        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$SERVER_PORT',port,STATUS) ;
        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$OPER',3,STATUS);
        SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$STATE',3,STATUS) ;

        CLOSE_FILE_(file_var,'S8:')
        DELAY 10
        OPEN_FILE_(file_var,'S8:')
        DELAY 10 ;
        CNV_STR_CONF('nut000', config_var, STATUS)
        n_bytes = 0
        delayCount = 0

connected::
 WRITE(CR)
```

```
BYTES_AHEAD (file_var, n_bytes, STATUS)

IF n_bytes = 0 THEN
        GOTO connected;
ENDIF

IF n_bytes > 3 THEN

        READ file_var (message:: 4) --Determine Command

        IF message = '0123' THEN
                DELAY 100
                GOTO connected;
        ENDIF

        IF message =  'A123' THEN
                WRITE('POZIVANJE PROGRAMA PICK - KUCISTE' ,CR)
                DELAY 100
                CALL_PROG ('SEMINAR_6',1)
                GOTO connected;
        ENDIF

        IF message = 'B123' THEN
                DELAY 100
                WRITE ('POZIVANJE PROGRAMA PICK - KRUG',CR)
                CALL_PROG ('PICK_2',1)
                GOTO connected;
        ENDIF

        IF message = 'C123' THEN
                DELAY 100
                WRITE ('POZIVANJE PROGRAMA PICK - POKLOPAC', CR)
                CALL_PROG ('PICK_1',1)
                GOTO connected;
        ENDIF

        IF message = 'D123' THEN
                DELAY 100
                WRITE ('POZIVANJE PROGRAMA DOD za poklopac', CR)
                CALL_PROG ('DOD_3',1)
                GOTO connected;
        ENDIF

        IF message = 'E123' THEN
```

```
                    DELAY 100
                    WRITE ('POZIVANJE PROGRAMA DOD za krug', CR)
                    CALL_PROG ('DOD_1',1)
                    GOTO connected;
                ENDIF

                IF message = 'F123' THEN
                    DELAY 100
                    WRITE ('POZIVANJE PROGRAMA DOD za kuciste', CR)
                    CALL_PROG ('DOD_2',1)
                    GOTO connected;
                ENDIF

                IF message = 'S123' THEN
                    DELAY 100
                    WRITE('STOP',CR)
                    GOTO izlaz;
                ENDIF

            ENDIF
izlaz::
        WRITE ('ZATVARAM FILE', CR)
        CLOSE_FILE_(file_var,'S8:')
END Kinect_10
```